



**Escola Politècnica Superior
de Castelldefels**

UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO DE FIN DE CARRERA

TÍTULO DEL TFC: Sensifire. Red de sensores móviles para la monitorización de incendios forestales

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones, especialidad Telemática

AUTORES: Norbert Nebra Riera
Santiago Pérez Fernández

DIRECTOR: Juan López Rubio

FECHA: 20 de enero de 2010

Título: Sensifire. Red de sensores móviles para la monitorización de incendios forestales

Autores: Norbert Nebra Riera
Santiago Pérez Fernández

Director: Juan López Rubio

Fecha: 20 de enero de 2010

Resumen

Este trabajo de fin de carrera presenta una versión inicial del sistema “Sensifire” que permite la monitorización de un escuadrón terrestre de bomberos. Este sistema se encarga de capturar, almacenar y representar las medidas de diferentes sensores incorporados al equipo del bombero.

El sistema permite la monitorización desde una estación central que representa geográficamente a cada uno de los bomberos junto las medias tomadas por su propio módulo de sensores. Además cada uno de los bomberos dispone de un dispositivo móvil en el que puede visualizar esta información.

El sistema presenta una arquitectura orientada a servicios (SOA) que permite que sea flexible, reutilizable, escalable y distribuido. La comunicación entre los diferentes servicios se realiza mediante el middleware MAREA (Middleware Architecture for Remote Embedded Applications) implementado por el grupo de investigación ICARUS de la escuela politécnica superior de Castelldefels.

A lo largo de esta memoria de proyecto se detallará el diseño del sistema y su implementación de hardware y software. Se ha requerido trabajar con conceptos de cartografía, sistemas empujados y móviles y bases de datos orientadas a objetos.

Title: Sensifire. Mobile sensors net for the monitoring of wildfires

Authors: Norbert Nebra Riera
Santiago Pérez Fernández

Director: Juan López Rubio

Date: January, 20th 2010

Overview

This project presents an initial version of the system “Sensifire” that allows the monitoring of a terrestrial firemen squad. This system is responsible for capturing, storing and represent the measurements from the different sensors that are included in the fireman equipment.

The system allows the monitoring from a main station that represents geographically each fireman with the measurements taken by its sensors module. Furthermore each fireman has a mobile device to view it.

The system presents a service-oriented architecture (SOA) allowing flexibility, reusability, scalable and distributed. The communication between all the services takes place using MAREA (Middleware Architecture for Remote Embedded Applications) implemented by the ICARUS research group, of the Politécnica Superior University, in Castelldefels, Spain.

Along this project report it is detailed the design of the system and its software and hardware implementation. It has been required working with cartography, embedded and mobile systems, and object-oriented database concepts.

ÍNDICE

INTRODUCCIÓN	1
CAPÍTULO 1. ARQUITECTURA DEL SISTEMA	4
1.1. Introducción	4
1.2. Arquitectura física	4
1.3. Arquitectura software	6
CAPÍTULO 2. SISTEMAS DE INFORMACIÓN GEOGRÁFICA (SIG)	8
2.1. Introducción	8
2.2. Conceptos previos	8
2.2.1. Sistemas de coordenadas	8
2.2.2. Datum	9
2.2.3. Ortofotografías	10
2.3. SharpMap	10
2.3.1. Peticiones a través del estándar WMS	11
2.3.2. Herramientas	12
2.3.3. Tablas de datos	13
2.3.4. Temas	14
2.3.5. Capas vectoriales	15
2.3.6. Capas de etiquetas	16
CAPÍTULO 3. SISTEMA DE ALMACENAMIENTO DE DATOS	18
3.1. Introducción	18
3.2. Requisitos del sistema	18
3.3. Análisis de alternativas	18
3.3.1. Perst	18
3.3.2. SQLite	19
3.3.3. SQL Server Compact	19
3.4. Elección	20
3.5. Diseño e implementación	20
3.5.1. Conceptos previos	20
3.5.2. Operaciones básicas	22
3.5.3. Diseño de la base de datos	24
CAPÍTULO 4: IMPLEMENTACIÓN DE HARDWARE	28
4.1. Introducción	28
4.2. Arduino	28

4.2.1.	Introducción	28
4.2.2.	Conexiones	29
4.2.3.	Entorno de programación	29
4.3.	Sensores.....	30
4.3.1.	Temperatura y Humedad.....	30
4.3.2.	Gas	36
4.3.3.	Otros gases	46
4.3.4.	Seguridad	48
4.4.	Conectividad Bluetooth	49
4.4.1.	Introducción	49
4.4.2.	Elección del módulo	49
4.4.3.	Configuración básica BT: comunicación con Arduino	50
4.5.	Consumo	52
4.5.1.	Solución a los problemas de consumo.....	53
4.6.	Esquema y montaje.....	53
 CAPÍTULO 5: MIGRACIÓN DEL MIDDLEWARE MAREA AL ENTORNO .NET COMPACT FRAMEWORK		56
5.1.	Introducción	56
5.2.	MAREA.....	56
5.3.	Migración al entorno .NET Compact Framework	58
5.3.1.	Introducción	58
5.3.2.	Estructura del código y archivos del proyecto.....	59
5.3.3.	Consola.....	60
5.3.4.	Librerías.....	62
5.3.5.	Cargador de servicios.....	63
5.3.6.	Traza y control de mensajes	64
 CAPÍTULO 6: IMPLEMENTACIÓN DISTRIBUIDA EN SERVICIOS.....		66
6.1.	Introducción	66
6.2.	Servicio: GPS	67
6.3.	Servicio: Arduino Sensors	69
6.4.	Servicio: Sensifire	70
6.5.	Servicio: Storage	72
6.6.	Servicio: Ground Station	73
6.6.1.	Introducción	73
6.6.2.	Conexión WMS.....	73
6.6.3.	Representación de datos.....	75
6.6.4.	Sistema de Alarmas	76
6.6.5.	Modos de funcionamiento	77
 CAPÍTULO 7: CONCLUSIONES		79
7.1.	Conclusiones tecnológicas	79

7.2. Impacto medioambiental	79
7.3. Conclusiones personales	80
7.4. Futuras líneas de trabajo	80
BIBLIOGRAFÍA	82
ANEXO 1. FUNCIONAMIENTO DEL SERVICIO GROUND STATION	84
ANEXO 2. MONTAJE DEL PROTOTIPO	90
ANEXO 3. INSTALACIÓN DRIVER MCL	92
ANEXO 4. CLASES	96
ANEXO 5. LIBRERÍAS EXTERNAS	98
A5.1. SharpMap	98
A5.2. SharpMap.UI	101
A5.3. Proj.NET	101
A5.4. Microsoft.WindowsMobile.Samples.Location	102
ANEXO 6. DATASHEETS	103
A6.1. Sensor de temperatura y humedad: SHT15	103
A6.2. Sensor de dióxido de carbono: CO2-D1	112
A6.3. Sensores de gases: MRQ1003-A y TGS 821	114
A6.4. Amplificador de instrumentación: INA126	117
A6.5. Módulo Bluetooth	124

INTRODUCCIÓN

Los incendios forestales son una preocupación que cada vez nos afecta más por todas sus consecuencias negativas, no sólo ambientales sino también sociales y económicas.

Son varias las consecuencias ambientales que se derivan de un incendio: destrucción de la masa vegetal, desaparición de ecosistemas, aumento de las emisiones de dióxido de carbono en la atmósfera, desertificación...

El efecto más fácilmente apreciable tras un incendio forestal es la pérdida de calidad paisajística debido a la destrucción de la cubierta vegetal, provocando un impacto paisajístico importante.

El efecto inmediato de los incendios forestales sobre la fauna es la muerte de aquellas especies que no pueden escapar del fuego. Por otro lado, la desaparición de los pastos y hábitats supone la migración de aquellas especies que logran sobrevivir.

Las condiciones climatológicas influyen en la susceptibilidad que un área determinada presenta frente al fuego; factores como la temperatura, la humedad y la pluviosidad determinan la velocidad y el grado al que se seca el material inflamable y, por tanto, la combustibilidad del bosque.

A las consecuencias ambientales de un incendio, hay que añadir toda una serie de implicaciones de orden económico más o menos cuantificables. Después de un incendio, se produce la pérdida de importantes recursos naturales directos e indirectos: madera, frutos, pastos, caza y pesca.

Los gastos necesarios para restaurar las zonas afectadas, así como las inversiones en prevención y extinción de incendios también suponen importantes partidas económicas.

El primer recurso que tenemos para defendernos de los incendios son las tareas de prevención. La conciencia social constituye una de las herramientas principales para la prevención de incendios. Mediante esta se debe educar a la población para hacer un uso racional del fuego, intentando reducir posibles situaciones de riesgo. No se debe olvidar que la mayor parte de los incendios forestales se deben a descuidos humanos o son provocados por el hombre.

También existen otras medidas de prevención como: el cuidado de las masas forestales mediante la realización de cortafuegos, la limpieza periódica de bosques, o la realización de quemas preventivas durante periodos de bajo riesgo de incendio. Todas estas medidas ayudan a reducir la velocidad de propagación de un incendio potencial.

Uno de los aspectos más importantes en la prevención es la detección precoz de los conatos de incendios forestales. Para ello es necesario un sistema que permita localizarlos antes de que tengan ocasión de extenderse. Las patrullas forestales con base en tierra y las torres de vigilancia han sido, en gran medida,

desplazadas por aeroplanos o helicópteros que detectan los incendios, determinan su localización en el mapa y vigilan su desarrollo.

Otro recurso del que se dispone en la lucha contra incendios son las tareas de extinción. Básicamente se utilizan dos estrategias: la mitigación del fuego y el control o vigilancia del incendio.

Tanto la detección temprana de incendios y vigilancia de los incendios durante su desarrollo son tareas cruciales para reducir el impacto negativo de los incendios forestales. La vigilancia de los incendios no sólo permite gestionar adecuadamente los recursos terrestres y aéreos, sino también permite salvar vidas humanas y bienes, además de obtener una alerta temprana sobre el comportamiento inesperado del incendio.

Existe también otro frente en la lucha contra los incendios forestales como la vigilancia de las brasas del fuego una vez este se extingue. Estas tareas se desempeñan en situaciones de visibilidad prácticamente nula debido al humo. Esto supone un riesgo para las vidas humanas que desempeñan dichas tareas.

Además de las consecuencias ambientales, los incendios, tienen una importante y negativa repercusión social. El trabajo de extinción de incendios forestales es una actividad de riesgo que todos los años es causa de accidentes mortales. El riesgo del personal que interviene en la extinción es, generalmente alto, como consecuencia del elevado número de incendios que se producen y, sobre todo, como consecuencia de las condiciones extremas en que se desarrolla el trabajo.

El proyecto Sensifire nace con el objetivo de prevenir y alertar al bombero enfrente de estas situaciones de riesgo mediante su monitorización en tiempo real.

Una de las ventajas de utilización de este sistema es que desde una única estación se pueden llegar a controlar la situación y condiciones de todos los escuadrones terrestres de bomberos. Para ello se ha diseñado un sistema que recoge automáticamente las medidas de diferentes sensores incorporados al equipo del bombero. De este modo se puede llegar a conocer el estado, situación y condiciones de cualquier bombero del escuadrón en cualquier instante.

Además el sistema permitirá hacer un análisis a posteriori de la actuación de estos para poder valorar la misma y corregir errores.

Para la implementación de una primera versión de este sistema se han marcado los siguientes objetivos:

- Crear un sistema portátil de sensores para poder incluirlo dentro del equipo de un bombero
- Implementar un sistema distribuido con una arquitectura SOA

(Arquitectura Orientada a Servicios) cuya comunicación se realizará a través del middleware MAREA.

- Utilizar tecnologías SIG (Sistemas de Información Geográfica) para controlar la cartografía y poder procesar los datos geográficos que necesita el sistema.
- Crear un sistema sencillo que permita monitorizar la situación del bombero en primera persona y remotamente desde una estación central.
- Utilizar un sistema de almacenamiento de datos.

El funcionamiento de este proyecto consiste en recoger las medidas de diferentes sensores que los bomberos llevan en su equipo. Posteriormente estas se muestran en un dispositivo móvil que posee el mismo y se envían a una estación central donde se guardan y se monitorizan.

La presente memoria está estructurada en siete capítulos principales: arquitectura del sistema, sistemas de información geográfica (SIG), sistema de almacenamiento, implementación de hardware, migración del middleware MAREA al entorno .NET Compact Framework y implementación distribuida en servicios.

CAPÍTULO 1. ARQUITECTURA DEL SISTEMA

1.1. Introducción

En este primer capítulo se presenta la arquitectura física y la arquitectura software del sistema. Se ha diseñado la arquitectura software para poder integrarse totalmente con la arquitectura física y se ha utilizado el middleware MAREA para poder realizar un sistema distribuido.

1.2. Arquitectura física

La arquitectura física del prototipo Sensifire está compuesta por dos partes diferenciadas: el hardware de la estación central y el del módulo del bombero.

El módulo del bombero está formado por una placa de sensores y una PDA que se comunican mediante *Bluetooth*. En la figura 1.1 se muestran los detalles de todos los componentes.

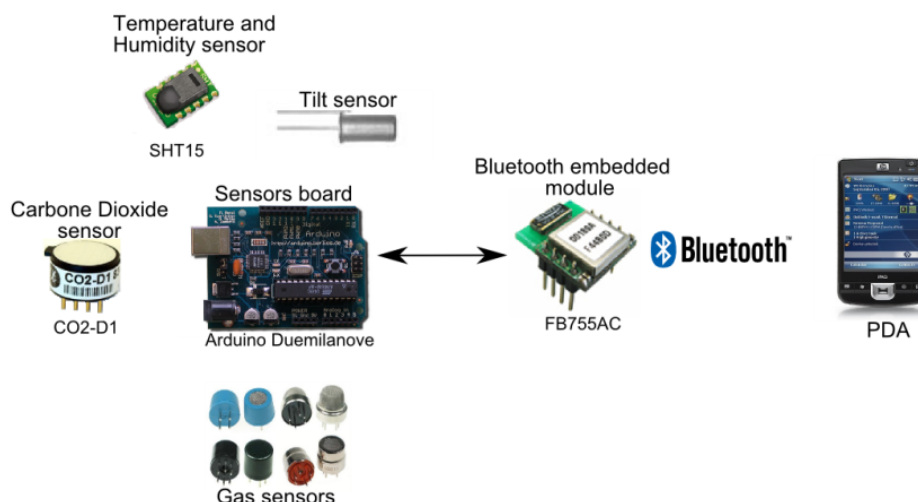


Fig. 1.1 Componentes del modulo del bombero

Cada uno de los módulos de los diferentes bomberos se comunicará con una estación central mediante una red descentralizada MANET (red móvil ad-hoc). Este tipo de redes son especialmente útiles en situaciones de emergencia, como desastres naturales o conflictos bélicos, al requerir muy poca configuración y permitir un despliegue rápido. Además cada uno de los nodos está preparado para reenviar datos de forma dinámica en función de la conectividad de la red.

En el escenario planteado se considera la utilización de múltiples redes MANET de modo que cada una de ellas disponga de un punto de acceso designado (camión de bomberos). En cada uno de los diferentes camiones se instalará: una antena omnidireccional WiFi para poder comunicarse con los bomberos y una direccional para poder establecer comunicación con los diferentes camiones. Esto se ha diseñado de este modo, ya que a priori las distancias a cubrir entre un camión y otro serán mucho más grandes que las existentes entre los bomberos y el camión.

La estación central se integrará en uno de los camiones de la flota del cuerpo de bomberos de Catalunya. En la figura 1.2 se muestran los componentes necesarios para su implementación.

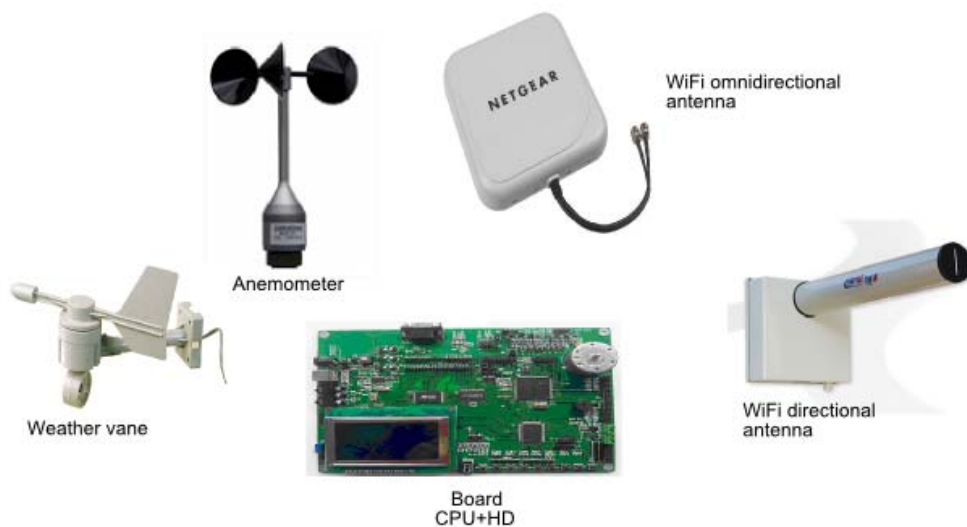


Fig. 1.2 Componentes de la estación central

A parte de instalar las dos antenas WiFi se incluirá un anemómetro y una veleta para poder evaluar el riesgo del incendio y obtener información sobre la propagación del fuego. El anemómetro y la veleta nos permitirán medir la fuerza y dirección del viento respectivamente. Para comprobar si las medidas obtenidas por estos son fiables se deberá incluir un inclinómetro para medir el ángulo de inclinación del camión.

Por otra parte, se precisará de la utilización de un PC para la gestión y monitorización de los bomberos mediante una interfaz gráfica.

En la siguiente figura se muestra un esquema general del escenario planteado:

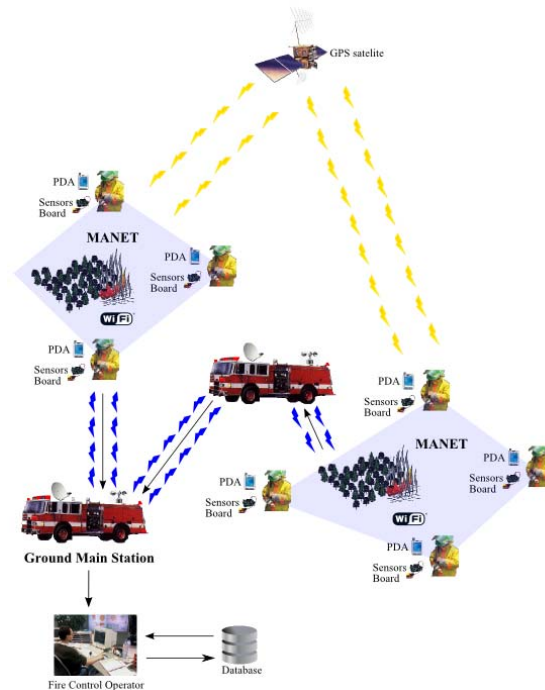


Fig. 1.3 Esenario general del sistema

1.3. Arquitectura software

Para la implementación de esta arquitectura se ha procedido a crear servicios distribuidos tal y como se muestra en la figura 1.4.

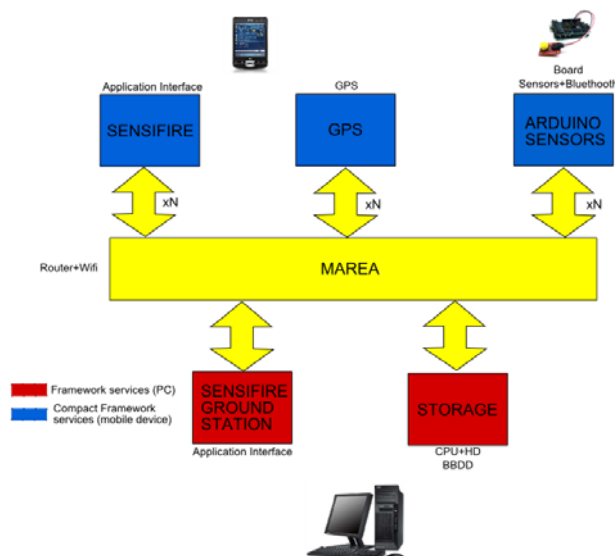


Fig. 1.4 Esquema de los componentes de la arquitectura software

Los servicios *Sensifire*, *GPS* y *Arduino Sensors* están diseñados para utilizarse en el entorno *Compact Framework*. Es decir, serán ejecutados en las PDA de cada uno de los diferentes bomberos. De este modo se tendrán tantos servicios de estos como bomberos registrados en el sistema.

El Servicio *GPS* se encarga de leer y procesar las sentencias NMEA que capta del receptor GPS de la PDA.

El componente *Arduino Sensors* lee las medidas de los diferentes sensores integrados en la placa *Arduino Duemilanove*.

Sensifire es la interfaz gráfica que muestra las medidas de cada uno de los sensores y la posición del bombero en la PDA.

El servicio *GroundStation* contiene la interfaz gráfica necesaria para la monitorización del escuadrón de bomberos. Este servicio será el encargado de comunicarse con el servidor de mapas para poder visualizarlos. El servicio ha sido diseñado con la idea de implementarse únicamente en la estación central aunque se podría ejecutar simultáneamente desde otro lugar. De esta manera se podría monitorizar al escuadrón en distintas máquinas a la vez.

El servicio *Storage* se utilizará para guardar las medias de los diferentes bomberos en una base de datos. Gracias a este, el servicio *GroundStation* podrá mostrar un histórico del recorrido del bombero y de las diferentes medidas de los sensores.

Todos los componentes distribuidos citados se han creado como servicios del middleware MAREA, que nos permite una gran flexibilidad a la hora de implementar este sistema. En el capítulo 6 se explicará la implementación de cada uno de los servicios.

CAPÍTULO 2. SISTEMAS DE INFORMACIÓN GEOGRÁFICA (SIG)

2.1. Introducción

Para la implementación del servicio *Ground Station* de la estación central es necesario manejar y analizar datos geográficos para poder representarlos sobre una cartografía. En el siguiente capítulo se introducen conceptos básicos de los sistemas de información geográfica. Además se analizan algunas de las funcionalidades de la librería SIG de programación *SharpMap*.

Un Sistema de Información Geográfica es una integración organizada de hardware, software y datos geográficos diseñado para capturar, almacenar, manipular, analizar y desplegar en todas sus formas la información geográficamente referenciada con el fin de resolver problemas complejos de planificación y gestión.

Los SIG son herramientas que permiten a los usuarios crear consultas interactivas, analizar la información espacial, editar datos, mapas y presentar los resultados de todas estas operaciones.

2.2. Conceptos previos

2.2.1. Sistemas de coordenadas

El sistema de coordenadas geográficas es el más utilizado y determina todas las posiciones de la superficie terrestre utilizando las dos coordenadas angulares de un sistema de coordenadas esféricas que está alineado con el eje de rotación de la Tierra.

La latitud mide el ángulo entre cualquier punto y el ecuador. Las líneas de latitud se llaman paralelos y son círculos paralelos al ecuador en la superficie de la Tierra.

La longitud mide el ángulo a lo largo del ecuador desde cualquier punto de la Tierra. Se acepta que Greenwich en Londres es la longitud 0 en la mayoría de las sociedades modernas. Las líneas de longitud son círculos máximos que pasan por los polos y se llaman meridianos.

Combinando estos dos ángulos, se puede expresar la posición de cualquier punto de la superficie de la Tierra como vemos en la figura 2.1.

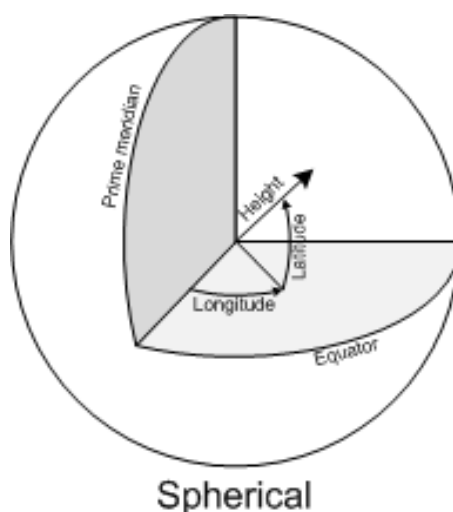


Fig. 2.1 Sistema de coordenadas geográficas

El proyecto utiliza coordenadas geográficas con datum WGS84. El principal motivo de su utilización es que el servicio GPS implementado utiliza este sistema.

El Sistema de Posicionamiento Global (GPS) fue creado por el Departamento de Defensa de los Estados Unidos. Este tuvo la necesidad de disponer de un sistema para posicionar una situación geográfica con referencia a un datum Universal con cobertura en toda la superficie terrestre, evitando así la territorialidad del resto de los datum existentes. Para ello fue creado el sistema WGS (World Geodetic System), estando actualmente vigente el WGS84 (ver [3], [4]).

La utilización de este sistema implica que los mapas utilizados en la interfaz gráfica SIG (servicio *GroundStation*) utilizarán el mismo. Si por otro lado se quisieran descargar los mapas de un servidor que utilizase otro sistema sería necesaria la conversión de coordenadas entre ellos.

2.2.2. Datum

La Tierra tiene forma parecida a un elipsoide regular (elipsoide de revolución), sin embargo no lo es, es un geoide, es decir un cuerpo levemente irregular y en consecuencia el saber con exactitud cuál es el centro geométrico del mismo no es un problema trivial.

Cualquier posición está referida a un modelo de Tierra, es decir a la elección de un centro de la Tierra. Esos modelos reciben el nombre de datum y deben ser

conocidos por quien utiliza el mapa. Cualquier punto georeferenciado debe constar de su posición de su altura y del datum utilizado. Los mismos datos pueden representar posiciones distintas según el datum utilizado y este error puede llegar a ser del orden de las centenas de metros (200 a 300 m).

Cada país utiliza en general un datum o modelo de la Tierra que se ajuste lo mejor posible a la forma de su terreno. El datum entonces está compuesto de un elipsoide teórico (esto implica la longitud del eje mayor y menor del mismo) y de un punto “fundamental” en que el elipsoide es tangente a la superficie de la Tierra (es decir que con el punto fundamental y los parámetros del elipsoide se está definiendo unívocamente la posición del centro de la Tierra). En el punto fundamental, las verticales de elipsoide y la Tierra coinciden (Fig. 2.2). También lo hacen las coordenadas del elipsoide y las de la Tierra.



Fig. 2.2 Punto Fundamental entre el geoide y el elipsoide

2.2.3. Ortofotografías

Los servidores de mapas utilizan ortofotos que son una presentación fotográfica de una zona de la superficie terrestre, en la que todos los elementos presentan la misma escala, libre de errores y deformaciones, con la misma validez de un plano cartográfico.

Una ortofotografía se consigue mediante un conjunto de imágenes aéreas (tomadas desde un avión o satélite) que han sido corregidas digitalmente para representar una proyección ortogonal sin efectos de perspectiva, y en la que por lo tanto es posible realizar mediciones exactas, al contrario que sobre una fotografía de aérea simple, que siempre presentará deformaciones causadas por la perspectiva desde la cámara, la altura o la velocidad a la que se mueve la cámara. A este proceso de corrección digital se le llama ortorectificación. Por lo tanto, una ortofotografía (u ortofoto) combina las características de detalle de una fotografía aérea con las propiedades geométricas de un plano.

2.3. SharpMap

SharpMap es una librería que facilita el trabajo con mapas. Está destinada para uso en web y aplicaciones de escritorio. El motor está escrito en C # y basado en .NET 2.0.

En el siguiente apartado se incluye una descripción de las funcionalidades básicas de esta librería, que son utilizadas para la implementación del servicio *GroundStation*.

2.3.1. Peticiones a través del estándar WMS

El protocolo de acceso utilizado para la descarga de mapas será el Web Map Service. El servicio Web Map Service (WMS) definido por el OGC (Open Geospatial Consortium) produce mapas de datos referenciados espacialmente, de forma dinámica a partir de información geográfica. Este estándar internacional define un "mapa" como una representación de la información geográfica en forma de un archivo de imagen digital para la exhibición en una pantalla de ordenador. Los mapas producidos por WMS se generan normalmente en un formato de imagen como PNG, GIF o JPEG, y ocasionalmente como gráficos vectoriales en formato SVG (Scalable Vector Graphics) o WebCGM (Web Computer Graphics Metafile).

Las operaciones WMS pueden ser invocadas usando un navegador estándar realizando peticiones en la forma de URL's (Uniform Resource Locators). El contenido de tales URL's depende de la operación solicitada. Concretamente, al solicitar un mapa, la URL indica qué información debe ser mostrada en el mapa, qué porción de la tierra debe dibujar, el sistema de coordenadas de referencia, y la anchura y la altura de la imagen de salida. Cuando dos o más mapas se producen con los mismos parámetros geográficos y tamaño de salida, los resultados se pueden solapar para producir un mapa compuesto. El uso de formatos de imagen que soportan fondos transparentes (GIF o PNG) permite que los mapas subyacentes sean visibles. Además, se pueden solicitar mapas individuales de diversos servidores.

Las operaciones WMS también pueden ser invocadas usando clientes avanzados SIG, realizando igualmente peticiones en la forma de URL's.

La librería *SharpMap* de código abierto que se utiliza en este proyecto, tiene la implementación integrada para realizar las funciones de cliente WMS, de tal manera que trabajar con los datos espaciales se hace eficiente.

SharpMap contiene la implementación de una capa WMS para poder realizar las peticiones y recuperar la información. El siguiente código muestra la carga de una capa WMS del *Institut Cartogràfic de Catalunya* (ICC).

```
WmsLayer WMSLayer = new WmsLayer( "WMSLayer",  
    "http://shagrat.icc.es/lizardtech/iserv/ows?" );  
  
WMSLayer.SpatialReferenceSystem = "EPSG:4326" ; ;  
WMSLayer.SetImageFormat( "image/jpeg" );  
WMSLayer.ContinueOnError = true;  
WMSLayer.Timeout = 10000;  
WMSLayer.AddLayer( "orto5m" );
```

Cuando se crea una capa WMS se le debe asignar un identificador y especificar la URL del servidor del que se tiene que descargar los mapas..

A la capa WMS definida en *SharpMap* se le pueden asociar diferentes atributos. Uno de los principales es el que indica la capa del mapa alojado en el servidor que se desea obtener. En el ejemplo, la capa *orto5m* corresponde a las imágenes ortográficas de Cataluña en un escala 1:5000. Además también se debe especificar el formato de de imagen en el que se proporcionará el mapa y el sistema de referencia espacial utilizado.

Para poder visualizar el mapa se debe añadir la capa WMS al objeto de de tipo *Map* de nuestro programa. Posteriormente este se debe asignar al control del tipo *MapImage* que muestra el Mapa. Tanto los objetos del tipo *Map* como *MapImage* son propios de la librería de *SharpMap*.

```
map.Layers.Add(WMSLayer);  
mapImageRE.Map=map;
```

Gracias a la utilización de *MapImage* se pueden obtener las coordenadas de cualquier punto del mapa a través de los eventos que impliquen realizar una acción con el ratón sobre el mapa. Este además ofrece la posibilidad de usar varios métodos específicos como actualizar la visualización de las capas por si alguna ha padecido alguna modificación, centrar el mapa en punto concreto, etc.

2.3.2. Herramientas

La librería *SharpMap.UI* ofrece un control propio del tipo *MapImage* para la representación de mapas. Uno de las ventajas de la utilización de este control es que ofrece herramientas para que el usuario pueda interactuar con el mapa.

El control implementa las siguientes funcionalidades: aumentar zoom, disminuir zoom, mover mapa y ampliar la zona seleccionada.

Lo único que se debe hacer para utilizar cada una de estas herramientas es asignarla como la activa.

```
//Aumentar Zoom  
mapImageRE.ActiveTool = MapImage.Tools.ZoomIn;  
//Disminuir Zoom  
mapImageRE.ActiveTool = MapImage.Tools.ZoomOut;  
//Mover el mapa  
mapImageRE.ActiveTool = MapImage.Tools.Pan;  
//Ampliar Zona  
map.ZoomToBox(Bbox);
```

La implementación de estas funciones se incluirá en el servicio *GroundStation* en forma de botones. De este modo, cada vez el usuario clique sobre uno de

ellos se activará una herramienta u otra. Posteriormente se podrá hacer uso de ella clicando encima del mapa.

2.3.3. Tablas de datos

SharpMap dispone del objeto del tipo *FeatureDataTable* para guardar información georeferenciada en una tabla. Este hereda de la librería *System.Data.DataTable*, por lo que su funcionamiento es muy similar.

En el proyecto se utilizaran estos objetos para guardar y representar las medidas de los diferentes bomberos en el mapa.

Una de las operaciones básicas de las tablas es su creación. A continuación se muestran los campos o columnas de una de las tablas utilizadas en el servicio *GroundStation*.

Tabla 2.1. Tabla de datos utilizada para guardar las medidas de temperatura y humedad tomadas por los sensores de un bombero

Identificador	Hora	Temperatura	Humedad	Tipo
Bombero1	12:23:50 12/12/09	34	80	weather

Para implementar una tabla como la anterior, se deben crear las columnas (objetos de tipo *DataColumn*). A cada una estas se le tienen que añadir un identificador y se debe especificar el tipo de datos que contendrán sus registros.

```
FeatureDataTable dtweather = new FeatureDataTable();

DataColumn Id = new DataColumn();
Id.DataType = System.Type.GetType("System.String");
Id.ColumnName = "Id";
dtweather.Columns.Add(Id);
```

Para poder añadir filas al *FeatureDataTable* se deben crear mediante el método *NewRow* que implementa dicho objeto.

```
FeatureDataRow rw = dtweather.NewRow();
rw.Geometry = new SharpMap.Geometries.Point(2.102, 41.35);

rw["Id"] = "Santi";
rw["Time"] = DateTime.Now;
rw["Type"] = "weather";
rw["Temperature"] = 28.3;
rw["Humidity"] = 87.23;
dtweather.AddRow(rw);
```

Uno de las particularidades de las filas *FeatureDataRow* de *SharpMap* es que permiten especificar coordenadas geográficas para cada una de las filas con el fin de dibujar cualquier tipo de geometría en el mapa (puntos, líneas y polígonos)

Si se desea representar una geometría en el mapa es necesario especificar a la capa correspondiente el *FeatureDataTable* del cual debe obtenerla.

```
LayerWeather.DataSource = new  
GeometryFeatureProvider(dtweather);
```

En el caso del servicio *GroundStation*, las geometrías dibujadas serán únicamente puntos representados por latitud y longitud en los que mostraremos un icono. Estos puntos corresponderán a la posición obtenida por el dispositivo GPS de cada uno de los bomberos.

2.3.4. Temas

La librería *SharpMap* contempla la creación de temas propios que se puedan aplicar en cualquier tipo capa y permitan la utilización de diferentes estilos dentro de cada una de ellas.

Para la implementación del servicio *GroundStation* se requiere la creación de un tema para las capas de tipo vectorial (2.3.6.). Gracias a esto se podrán dibujar símbolos diferentes en cada una de las capas.

Para crear un tema para las capas de tipo vectorial se debe crear un objeto del tipo *CustomTheme* al que se le pasee en su constructor un método que devuelva un objeto del tipo *VectorStyle*.

```
CustomTheme myTheme = new CustomTheme(GetStyle);

private VectorStyle GetStyle(FeatureDataRow row)
{
    VectorStyle style = new VectorStyle();

    switch (row["Type"].ToString().ToLower())
    {
        case "weather":
            Bitmap w = new Bitmap(Path.Combine(path,
                @"temperature3.gif"));
            style.Symbol = w;
            style.SymbolScale = 1;
            return style;
        case "gas":
            Bitmap g = new Bitmap(Path.Combine(path,
                @"gas3.gif"));
            style.Symbol = g;
            return style;
        default:
            return style;
    }
}
```

Para poder dibujar símbolos diferentes, se debe añadir una columna a la tabla que permita escoger el estilo que se desea aplicar (columna “Tipo” de la tabla 2.1). En función del valor de los registros de esta columna se dibujará un icono u otro según la implementación del método GetStyle.

2.3.5. Capas vectoriales

VectorLayer es un tipo de capa propia de *SharpMap* que como su nombre indica contiene datos del tipo vectorial.

Las capas del tipo vectorial se utilizan para la representación de tres tipos geometrías básicas:

- Puntos: Los puntos se reducen a pares de coordenadas latitud-longitud o x-y que marcan la posición de lo modelizado sobre la superficie de la tierra. En este proyecto se utilizarán para representar la posición del bombero.
- Líneas: Son una serie ordenada de posiciones unidas por segmentos rectos. Este tipo se utilizará para mostrar la ruta que ha seguido el bombero.
- Polígonos: Son líneas cerradas que delimitan superficies.

Para dibujar una línea es necesario seguir los siguientes pasos:

1. Crear una lista con los puntos que la formaran. Se debe tener en cuenta el orden en que se añaden estos ya que estos se dibujaran uniendolos cada uno de ellos con su anterior respectivo.

```
List<Point> Points = new List<Point>();
Points.Add(new Point(0,0));
Points.Add(new Point(0,1));
```

2. Crear un objeto del tipo *LinearString* de *SharpMap* con la lista de puntos anteriormente creada. Con esto se consigue crear la línea que pase por todos los puntos.

```
LinearString line = new LinearString(Points);
```

3. Añadir la línea a una colección del tipo *Geometry*. En este caso la geometría corresponde a una línea, es decir un objeto del tipo *LinearString*.

```
Collection<Geometry> lineCol = new
Collection<Geometry>();
lineCol.Add(line);
```

4. Para acabar se debe especificar a la capa, del mismo modo que se hace en el caso de los puntos, de donde debe obtener las geometrías para dibujarlas. En este caso se toman de la colección del tipo *Geometries* creada anteriormente.

```
VectorLayer LayerLines=new LayerLines("Lines Layer");
LayerLines.DataSource=new GeometryProvider(lineCol);
```

2.3.6. Capas de etiquetas

SharpMap dispone de una capa de etiquetas para mostrar cadenas de texto. Esta capa se utilizará para identificar y asociar cada uno de los iconos mostrados en el mapa con el bombero correspondiente.

Para crear una capa de etiquetas es necesario crear un objeto del tipo *LabelLayer*. Del mismo modo que se hace en las capas vectoriales, se debe especificar el origen de los datos, para posteriormente dibujarlos. En este caso la información se obtiene de la geometría almacenada en cada una de las filas de la tabla *dtgas*.

Las cadenas de texto que se dibujaran para cada una de estas geometrías (puntos) se obtienen de la columna *Id* de esta tabla. Los registros de esta columna contienen los identificadores de los diferentes bomberos.

Las capas de etiquetas permiten definir un estilo para la presentación del texto. El estilo nos permite escoger el tipo, tamaño, alineación y color del texto utilizado.

```
LabelLayer LayerLabelGas = new LabelLayer("Label Layer  
Gas");  
LayerLabelGas.DataSource = new  
GeometryFeatureProvider(dtgas);  
LayerLabelGas.LabelColumn = "Id";  
LayerLabelGas.Style = new LabelStyle();  
LayerLabelGas.Style.ForeColor = Color.Black;  
LayerLabelGas.Style.Font = new  
Font(FontFamily.GenericSerif, 11);  
LayerLabelGas.Style.HorizontalAlignment=LabelStyle.Horizon  
talAlignmentEnum.Left;  
LayerLabelGas.Style.VerticalAlignment =  
LabelStyle.VerticalAlignmentEnum.Bottom;  
LayerLabelGas.Style.Offset = new PointF(3, 3);  
LayerLabelGas.Style.Halo = new Pen(Color.Yellow, 2);  
LayerLabelGas.TextRenderingHint =  
System.Drawing.Text.TextRenderingHint.AntiAlias;  
sharpMap.Layers.Add(LayerLabelGas);
```

CAPÍTULO 3. SISTEMA DE ALMACENAMIENTO DE DATOS

3.1. Introducción

En el siguiente capítulo se presentan los requisitos que debe cumplir el sistema de almacenamiento de datos. Posteriormente se analizan de forma breve diferentes alternativas disponibles. El apartado concluye con la elección de una de estas y el diseño utilizado para la implementación del proyecto.

3.2. Requisitos del sistema

Compatible con .NET: Uno de los requisitos del proyecto es la implementación de este en .NET utilizando el lenguaje de programación C#, por lo tanto la base de datos seleccionada tiene que ser compatible con este entorno y lenguaje.

Multiplataforma: En el proyecto se ha hecho indispensable la carga/guardado de datos tanto desde un dispositivo móvil (PDA) como aplicación de escritorio (interfaz SIG).

Rapidez: Aunque el volumen de datos de esta aplicación no sea grande y las estructuras de datos que se deben guardar no son complejas es necesario que el acceso a estos sea lo más rápido posible.

Fiabilidad: Los datos que guarda nuestra aplicación son sumamente importantes, ya que de estos dependen posibles intervenciones que pueden llegar a evitar y solventar situaciones de peligro con las que se encuentren los bomberos. Por este motivo es básico garantizar la fiabilidad todas las transacciones que se hagan en la base de datos.

3.3. Análisis de alternativas

3.3.1. Perst

Perst es una base de datos embebida orientada a objetos para *Java*, *.NET* y *Mono*. El término “orientación a objetos” en este caso se refiere a que *Perst* carga y guarda los objetos directamente. El término embebida significa que se utiliza en aplicaciones que requieren su propio almacenamiento de datos y que no utiliza una aplicación o servidor externo para funcionar.

Perst guarda la información en proyectos *Java* o *.NET* eliminando la translación requerida para el almacenamiento en una base de datos de tipo relacional (ver [10]).

El código fuente de núcleo de este no supera las 500 líneas es abierto y accesible. El programador obtiene un control total de su aplicación y la interacción con la base de datos. Además este requiere un uso mínimo de los recursos del sistema.

La rapidez es una de sus características más destacables en frente de otras bases de datos embebidas orientadas a objetos de *Java* y *.NET*.

Perst soporta transacciones con las propiedades *ACID*. En las bases de datos, *ACID* son conjunto de características necesarias para que una serie de instrucciones pueda ser considerada una transacción (atomicidad, consistencia, aislamiento y durabilidad). Además no requiere de la administración por parte del usuario final.

Perst se distingue por su facilidad para trabajar con objetos y está diseñado para trabajar con herramientas como *AspectJ* (extensión de *Java*) y *JAssist*. Además no es necesaria la utilización de preprocesadores, compiladores o máquinas virtuales para su utilización.

La API de *Perst* es flexible y fácil de utilizar. Esta incluye índices para la representación espacial, contenedores para la optimización de memoria, un contenedor que ofrece una interfaz similar a SQL (*SubSQL*), importación y exportación de XML, replicación de datos...

3.3.2. SQLite

Es un sistema de gestión de bases de datos relacional compatible con *ACID*, y que está contenida en una pequeña librería (500 kB). Es un proyecto de código abierto creado por D. Richard Hipp.

Esta librería se enlaza con el programa pasando a formar parte integral del mismo. El programa utiliza la funcionalidad de *SQLite* a través de llamadas a subrutinas y funciones. Esto reduce la latencia en comparación a la comunicación entre procesos de otras bases de datos.

Es multiplataforma y cumple con la mayoría de los estándares *SQL92*. Por lo que no posee curva de aprendizaje si se conoce *SQL* o *MySQL*.

3.3.3. SQL Server Compact

Es un motor de base de datos relacional, de libre descarga y distribución, tanto para móviles como para aplicaciones de escritorio. Está orientada a sistemas ocasionalmente conectados, ofrece características especialmente útiles para

clientes ligeros. Desde la versión 2.0, el lanzamiento de *SQL Server Compact* ha ido ligado al de *Microsoft Visual Studio .NET*.

3.4. Elección

Una de las grandes diferencias entre las diferentes alternativas es que *SQLite* y *SQL Server Compact* son bases de datos de tipo relacional a diferencia de *Perst* que está orientada a objetos. Este ha sido uno de los factores fundamentales que ha hecho que nos decanemos por esta última.

El hecho de utilizar un lenguaje de programación orientado a objetos como *C#* plantea como solución más sencilla y directa la carga y guardado de estos objetos directamente en la base de datos, cosa que no se hace en bases de datos de tipo relacional.

Por otra parte tal y como se comenta en los requisitos, la base de datos necesaria para un proyecto de este tipo, a priori, no tendría porque presentar una estructura de datos demasiado compleja, con muchas tablas o relaciones. Por este motivo la utilización de una base de datos de tipo relacional se hace innecesaria.

3.5. Diseño e implementación

3.5.1. Conceptos previos

3.5.1.1. Bases de datos orientadas a objetos

Para un programa resulta natural trabajar con objetos. Sin embargo, esto es imposible para las bases de datos tradicionales, basadas en el modelo relacional. Para resolver este problema aparecieron las bases de datos orientadas a objetos, como *Perst*. Al contrario de las relacionales, que trabajan con registros, las bases de datos orientadas a objetos guardan y recuperan objetos. Por lo tanto, la comunicación de este tipo de base de datos con un programa orientado a objetos es transparente (ver [9]).

Aunque sobre el papel, está es la mejor opción para implementar una aplicación de base de datos, en la práctica presenta una serie de problemas, debido a las características actuales de las bases de datos orientadas a objetos. Éstas no están tan maduras como las relacionales y por lo tanto no gozan de la abundancia de herramientas, fiabilidad, y rendimiento de estas últimas.

Uno de los problemas de estas bases de datos es su incompatibilidad entre ellas. Esto hace imposible migrar una base de datos orientada a objetos a otra, lo que nos obliga a depender de un único proveedor (“vendor lock-in”), con

todos los inconvenientes que esto representa (obligación a aceptar aumentos de precio, falta de soporte si el proveedor sale del mercado ...).

3.5.1.2. *Persistencia transparente*

Esta característica nos permite que la aplicación trate con objetos persistentes utilizando una API orientada a objetos sin la necesidad de lenguaje SQL embebido en el código. Los objetos son recuperados y almacenados por el sistema cuando éste lo cree conveniente, sin que el usuario deba hacer ninguna solicitud explícita de consulta, actualización o recuperación de información de una base de datos orientada a objetos.

Para poder guardar objetos en una base de datos Perst, es necesario otorgar persistencia a los objetos que se deseen almacenar. Esto se consigue haciendo que las clases de dichos objetos hereden de la clase *Perst.Persistent*.

```
public class User : Persistent
{
    [Indexable(Unique = true)]
    public String Id;

    [Indexable]
    public String Name;

    public override String ToString()
    {
        return Name + " (" + Id + ")";
    }
}
```

3.5.1.3. *LINQ*

Language Integrated Query (LINQ) es un proyecto de Microsoft que agrega consultas nativas semejantes a las de SQL a los lenguajes de la plataforma .NET, inicialmente a los lenguajes *Visual Basic .NET* y *C#*. Muchos conceptos que LINQ ha introducido fueron originalmente probados en Cw, un proyecto de investigación de Microsoft.

LINQ define *operadores de consulta estándar* que permiten a lenguajes habilitados con LINQ filtrar, enumerar y crear proyecciones de varios tipos de colecciones usando la misma sintaxis. Tales colecciones pueden incluir vectores, clases enumerables, XML, conjuntos de datos desde bases de datos relacionales y orígenes de datos de terceros. El proyecto LINQ usa características de la versión 2.0 del .NET Framework, nuevos ensamblados relacionados con LINQ, y extensiones para los lenguajes C# y Visual Basic .NET. Microsoft ha distribuido una versión previa del LINQ, consistente de estas bibliotecas y compiladores para C# 3.0 y Visual Basic 9. En la base de datos de este sistema utilizaremos LINQ para consultar las medidas de los diferentes bomberos en relación a la posición o instante en que estas fueran tomadas.

3.5.1.4. Emulación de tablas

Unos de puntos que tiene a favor el trabajar con objetos persistentes respecto a los normales o transitorios son: la mejora de eficiencia para tareas de mantenimiento de clases extendidas, bloqueo de objetos, comparativa entre instancias a objetos para detectar campos actualizados, actualización de índices, optimización, procesado, análisis de consultas...

Por el contra existen tareas que suponen un trabajo extra para el programador como el mantenimiento de los índices, control de bloqueos y escribir código para implementar consultas.

Este trabajo se puede evitar utilizando la clase *Database* de Perst, que proporciona una API similar a la utilizada en las bases de datos de tipo relacional.

Esta clase proporciona las siguientes funcionalidades (ver [11]):

- Mantiene las clases extendidas (tablas) que permiten iterar a través de todas las instancias de una clase. No es necesario definir un objeto root en la base de datos.
- Proporciona transacciones serializables utilizando el bloqueo de tablas.
- Proporciona el lenguaje de consultas JSQL, un subtipo de lenguaje SQL orientado a objetos aunque no soporta joins ni las funciones grouping y aggregate.
- Mantenimiento de los índices: El programador crea los índices implícitamente como atributos y estos son creados y añadidos explícitamente por la base de datos cuando se añade un objeto en ella. Cuando un objeto se elimina de la base de datos esta también lo remueve de los índices. El único inconveniente de esto es que las actualizaciones deben ser controladas por el programador. Un objeto se debe de eliminar del índice antes de actualizarse y se debe añadir en el después de la actualización.

En este sistema se utilizará la emulación de tablas con el fin de reducir la complejidad de gestión de la base de datos implementada.

3.5.2. Operaciones básicas

En este apartado se incluyen las operaciones básicas utilizadas para poder guardar y obtener datos de una base de datos *Perst*. Comentar que los modelos introducidos a continuación no son los únicos que implementa la API de *Perst*.

3.5.2.1. Almacenamiento de objetos

Uno de los primeros objetivos de *Perst* es permitir al programador trabajar con objetos persistentes, en la medida de lo posible, de la misma manera que se trabaja con objetos transitorios. Sin embargo, esta persistencia posee algunos aspectos no transparentes para el programador. Uno de los ejemplos de esto se encuentra en el almacenamiento.

El fichero donde se almacenan los objetos debe ser especificado por el programador mediante la clase *Storage*. Esta clase es abstracta por lo que se debe usar la clase *StorageFactory* para crear una instancia del tipo *Storage*.

Una vez se crea la instancia se debe abrir el fichero que contiene la base de datos mediante el método *Open*.

```
Storage storage = StorageFactory.Instance.CreateStorage();  
storage.Open("ddbbs.dbs");
```

En vistas a reducir la complejidad de la gestión de la base de datos se utilizará la emulación de tablas. Para poderla utilizar se debe crear un objeto del tipo *Database* al que se le pase, en su constructor, el objeto *storage* creado anteriormente.

```
Database db = new Database(storage);
```

Para añadir un objeto a la base de datos mediante la emulación de tablas es necesario que el objeto que desee añadir herede de la clase *Persistent*. Para guardarlo se debe invocar al método *AddRecord* del objeto del tipo *Database*. Este objeto insertará automáticamente el objeto en los índices.

Para cerrar la base de datos correctamente se debe invocar al método *Close* de la clase *Storage*.

```
storage.Close();
```

3.5.2.2. Índices

Para crear índices automáticamente en los objetos que se añadan a la base de datos se debe marcar los campos de la clase que se quieran indexar mediante el atributo *Indexable*.

```
public class User : Persistent
{
    [Indexable(Unique = true)]
    public String Id;

    [Indexable]
    public String Name;

    public override String ToString()
    {
        return Name + " (" + Id + ")";
    }
}
```

3.5.2.3. Obtención de objetos

Para obtener objetos de la base de datos se utilizará las consultas mediante el lenguaje *LINQ*. Para ello es necesario utilizar la librería *PerstNET20.dll* para .NET Framework 3.5. Esta es la única versión de *Perst* que soporta la utilización de este lenguaje.

Esta versión incluye el método *GetTable<>* dentro de la clase *Database* (emulación de tablas), que permitirá obtener una tabla que incluya todos los objetos de un tipo determinado que se encuentran dentro de la base de datos.

La siguiente consulta retornaría una lista todos los identificadores de los objetos de tipo *User* que se encuentran en la base de datos.

```
public List<string> Get_Users()
{
    var query = from u in db.GetTable<User>()
    select u.Id;

    return query.ToList();
}
```

3.5.3. Diseño de la base de datos

3.5.3.1. Diseño

La base de datos dispone cinco tablas diferentes para el almacenamiento de los datos. Cuatro de estas cinco tablas se utilizan para guardar los diferentes tipos de media (gas, posición, medidas ambientales y seguridad). Cada una de ellas almacena además de las propias medidas, una referencia temporal de cuando se ha tomado la medida y un identificador para diferenciar el bombero

que provienen dichas medidas. Gracias a esto podremos saber dónde, cuando y por que bombero se toman cada una de las diferentes medidas.

La quinta tabla se utiliza para mantener un registro de los bomberos que hay o ha habido conectados al sistema. Esta tabla se gestiona de manera autónoma. Cada vez que llega una medida al resto de tablas, las de medidas, se comprueba si existe una referencia de que ese bombero este registrado en el sistema. En caso que no lo este, se añade el identificador del bombero a esta tabla. En caso contrario no se hace nada.

Todas las tablas son transparentes para el programador y solo se las llamara en el caso que se quiera hacer una consulta a la base de datos mediante *LINQ* y el método *GetTable<>* de *Perst*. Los resultados a las consultas se retornaran en objetos del tipo *GasPosition* y *WeatherPosition*, que permiten referenciar cada una de las medidas (gas o ambiental) con la posición en la que fueron tomadas.

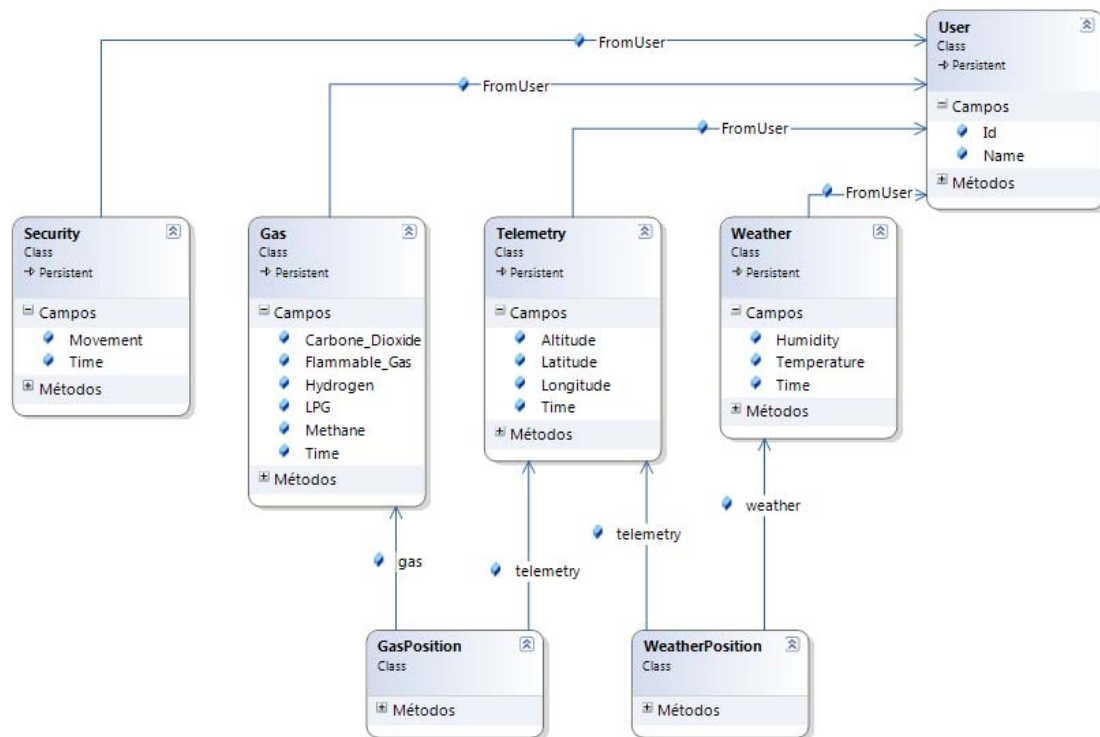


Fig. 3.1 Clases utilizadas para la implementación de la base de datos Perst

3.5.3.2. Almacenamiento de medidas

Para guardar las medidas se han creado cuatro funciones diferentes según el tipo de medida que se desea guardar. Comentar que todas ellas siguen una estructura similar. La única diferencia existente es el tipo de objeto que guardan.

El siguiente código muestra como se guarda una medida de gas. El método *UpdateUsers* es el que se encarga de consultar si el bombero del que proviene la medida está registrado en la tabla de bomberos.

```
public void AddGas(String Id, String Name, double
Flammable_Gas, double Hydrogen, double Carbone_Dioxide,
double Methane, double LPG, DateTime Time)
{
    Gas g = new Gas();
    g.Flammable_Gas=Flammable_Gas;
    g.Hydrogen=Hydrogen;
    g.LPG=LPG;
    g.Methane=Methane;
    g.Carbone_Dioxide=Carbone_Dioxide;
    g.Time = Time;
    g.FromUser = UpdateUsers(Id, Name);

    db.AddRecord(g);
}
```

3.5.3.3. Obtención de medidas

Este proyecto esta implementado de forma distribuida de manera que los datos se generan, se envían y se guardan de manera separada. Uno de los propósitos principales del proyecto es monitorizar el estado de los bomberos, mediante una interfaz gráfica SIG. Para hacer esto se hace indispensable, relacionar de alguna manera esta información para poderla representar. Por ejemplo, para representar las medidas de un bombero en un mapa necesitaremos relacionar estas con una posición tomada por el dispositivo GPS en un instante más o menos próximo, dependiendo de la exactitud que se le quiera imponer al sistema.

Mediante LINQ se llevará a cabo dicha tarea en la misma base de datos Perst. Esto se podría haber hecho en la interfaz gráfica (servicio *GroundStation*), pero se debe intentar minimizar el funcionamiento de la interfaz gráfica SIG con el fin de hacerla lo más rápida y fluida posible.

Para ello las funciones utilizadas para hacer las consultas deberán retornar objetos que contengan además de la media, una referencia geográfica de donde fue tomada. El siguiente código muestra una de las clases utilizadas para representar las medidas ambientales.

```
public class WeatherPosition
{
    public Position position;
    public Weather weather;
    public WeatherPosition(Weather weather, Position
position);
}
```

La base de datos se ha diseñado de manera que se puedan hacer consultas en función de dos parámetros, el tiempo o la posición. Es decir se podrán buscar la medidas de un bombero comprendidas entre dos intervalos de tiempo o las que se encuentran dentro de un intervalo de coordenadas.

Además solo se podrán consultar las medidas de un único bombero a la vez. Esto se ha hecho de esta manera básicamente pensando en la representación que se hará de estas en la estación central (*GroundStation*). El hecho de dibujar las rutas que siguen todos bomberos a la vez puede dificultar la visualización de estas.

En el código siguiente se muestra una consulta mediante LINQ que busca las medidas ambientales (*weather*) de un bombero con identificador *Id* entre dos intervalos de tiempo (*Max,Min*) en relación a su posición.

```
var query = from w in db.GetTable<Weather>()
join t in db.GetTable<Telemetry>() on w.FromUser equals
t.FromUser
where w.FromUser.Id == Id && t.FromUser.Id == Id && w.Time
> Min && w.Time < Max && t.Time > Min && t.Time < Max &&
w.Time.Minute==t.Time.Minute
orderby w.Time
select new WeatherPosition( w.weather(), t.telemetry() );
```

Para hacer esta consulta se utiliza la sentencia *join* para combinar los objetos *Weather* y *Telemetry* de sus respectivas tablas, que provienen de un mismo bombero con identificador *Id*.

En la sentencia *where* se filtran solamente aquellos objetos que tengan una referencia de tiempo entre los intervalos *Max* y *Min*. Además para que el resultado de esta consulta sea correcto debemos especificar que la medida ambiental y la posición con que se relaciona se encuentren en un mismo instante. Concretamente lo que se hace es relacionar solamente aquellas que se hayan tomado en un mismo minuto. El resultado de esta consulta se retorna en un objeto del tipo *WeatherPosition*.

3.5.3.4. Mantenimiento de los índices

El diseño implementado de la base de datos no contempla la actualización de los objetos guardados en ella. Lo único que se hace en la base de datos es añadir datos y hacer consultas. Por este motivo no es necesario el mantenimiento de estos, tal y como se explica en el apartado 3.5.1.2.

CAPÍTULO 4: IMPLEMENTACIÓN DE HARDWARE

4.1. Introducción

Además de la estación central, el sistema esta compuesto por diferentes módulos que llevan los bomberos. Estos módulos recogen información de diferentes medidas (posicionamiento GPS, CO₂, temperatura, humedad). La información se muestra en la pantalla del dispositivo PDA, para realizar esto se requiere de un hardware diseñado específicamente para ello. En este capítulo de describen su diseño y características más importantes.

4.2. *Arduino*

4.2.1. Introducción

Para realizar una buena implementación hardware se requiere un controlador, para poder manejar los dispositivos que conectemos, ya sean I²C, ya sean comunicaciones tipo UART y para leer valores de los sensores, poderlos interpretar y manipular para que lleguen a su destino.

Se observan las diferentes alternativas y la familia de microcontroladores ATMEL parece la más fiable, integrados en nuestro caso dentro de la placa Arduino detallada a continuación.

Ante la necesidad de usar un microcontrolador para leer las medidas que nos devuelven los sensores, se ha decidido usar la placa “Arduino”. Al ser open-hardware, su diseño y distribución es libre, por lo tanto no se requiere ningún tipo de licencia para usarlo.

Arduino es una placa basada en microcontroladores Atmega168 o Atmega328 (chips sencillos y de bajo coste que permiten desarrollar múltiples diseños) y en sencillez de conexiones. Arduino es una interface hardware libre y simple con lenguaje de programación Processing/Wiring (ver [13]).

4.2.2. Conexiones

El conexionado de la placa se muestra en la siguiente figura:

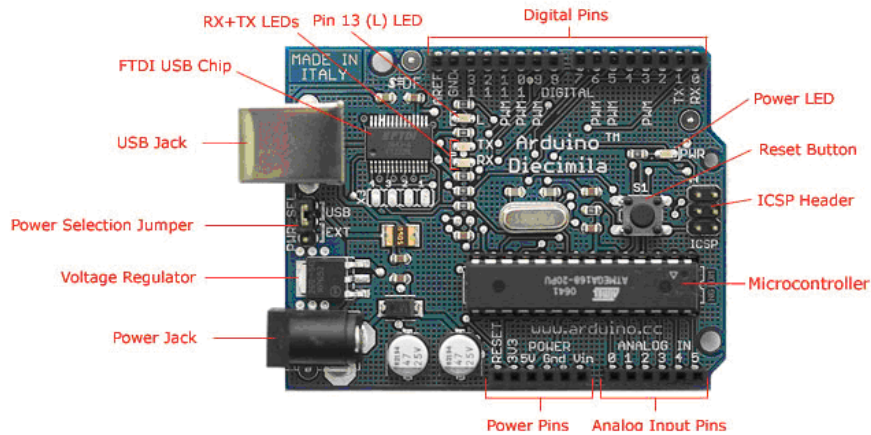


Fig. 4.1 Esquema de conexiones de Arduino

Se alimenta usando la entrada USB (que funciona a 5V) o mediante un conector jack de alimentación (entre 7 y 10V recomendado, límites entre 6 y 20V).

Incorpora un regulador de tensión, que nos entrega salidas a 3.3 y 5V, muy útiles para realizar montajes.

Los pines “analog in” realizan una digitalización de la señal analógica pasándola de 0-5V a un valor entre 0 y 1024 (10 bits de resolución).

Los 14 pines digitales se pueden usar como entrada o salida usando valores lógicos (0 a 0V y 1 a 5V). Los pines 3, 5, 6, 8, 10 y 11 pueden proporcionar una salida PWM (Pulse Width Modulation). Si se conecta cualquier componente a los pines 0 y 1 (TX y RX), eso interferirá con la comunicación USB, debido a que estos pines están conectados al chip para la comunicación serie FTDI USB-to-TTL.

4.2.3. Entorno de programación

La placa puede ser programada con el software propio de Arduino figura 4.2 basado en librerías C++ y también puede ser programado desde Arduino con lenguaje de programación AVR C.

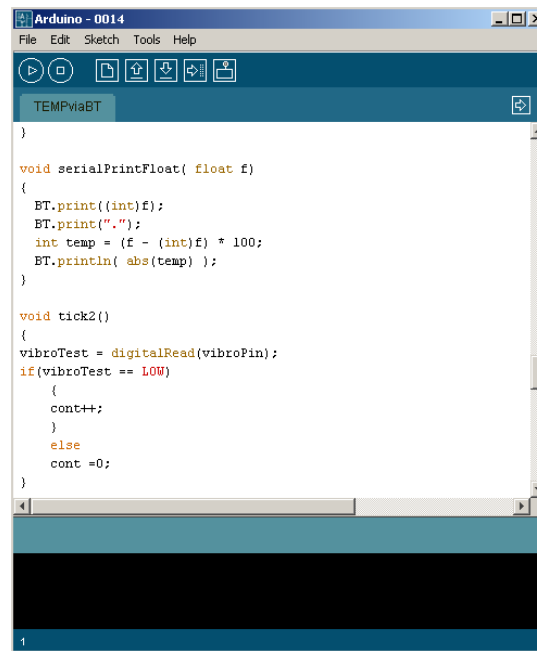


Fig. 4.2 Entorno de programación Arduino

4.3. Sensores

4.3.1. Temperatura y Humedad

4.3.1.1. Elección del sensor de temperatura y humedad

Se han comparado diferentes soluciones para obtener mediciones de temperatura y humedad, mostrados en la siguiente tabla comparativa.

Tabla 4.1. Comparativa de dispositivos

		SHT15	DS18B20	MCP9700	808H5V5
Resolution	Temperatura	0,01° C	0,5 to 0,0625° C	=====	=====
	Humedad	0,05%RH	=====	=====	=====
Accuracy	Temperatura	±0,3° C	±0,5° C	±2° C	=====
	Humedad	±2%RH	=====	=====	±4%
Response time	Temperatura	8s	750ms	1,3s	=====
	Humedad	5-30s	=====	=====	15s
Operating range	Temperatura	-40 to 123,8° C	-55 to 125° C	-40 to 125° C	=====
	Humedad	0-100%RH	=====	=====	-40 to 85°C
Price		31,15€	4,25\$	2,50€	15€

Para la captación de parámetros meteorológicos (temperatura y humedad) se utiliza el sensor SHT15 (Fig. 4.3), debido a la integración de ambos sensores, su facilidad de comunicación (sin circuito de adaptación) y resulta fácil de conseguir. Las características principales se listan a continuación:

- Totalmente calibrado.
- Salida digital I²C.
- Bajo consumo de potencia (80μW (at 12bit, 3V, 1 measurement/s)).
- Rango de temperatura: de -40° C a 125° C.
- Rango de humedad: entre 0 y 100%(humedad relativa).
- Estable durante un funcionamiento prolongado.
- Los límites de la precisión máxima de humedad relativa y temperatura se muestran en las figuras 4.4 y 4.5 respectivamente obtenidas del datasheet del fabricante.



Fig. 4.3 Sensor de humedad y temperatura SHT15

Relative Humidity

Parameter	Condition	min	typ	max	Units
Resolution ¹		0.4	0.05	0.05	%RH
		8	12	12	bit
Accuracy ² SHT10	typical		±4.5		%RH
	maximal	see Figure 2			
Accuracy ² SHT11	typical		±3.0		%RH
	maximal	see Figure 2			
Accuracy ² SHT15	typical		±2.0		%RH
	maximal	see Figure 2			
Repeatability			±0.1		%RH
Replacement	fully interchangeable				
Hysteresis			±1		%RH
Nonlinearity	raw data		±3		%RH
	linearized		<<1		%RH
Response time ³	τ (63%)		8		s
Operating Range		0		100	%RH
Long term drift ⁴	normal		< 0.5		%RH/yr

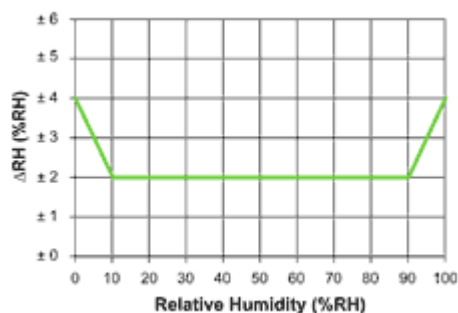


Fig. 4.4 Parámetros del sensor (humedad) y curva de precisión

Temperature

Parameter	Condition	min	typ	max	Units
Resolution ¹		0.04	0.01	0.01	°C
		12	14	14	bit
Accuracy ² SHT10	typical		±0.5		°C
	maximal	see Figure 3			
Accuracy ² SHT11	typical		±0.4		°C
	maximal	see Figure 3			
Accuracy ² SHT15	typical		±0.3		°C
	maximal	see Figure 3			
Repeatability			±0.1		°C
Replacement		fully interchangeable			
Operating Range		-40		123.8	°C
		-40		254.9	°F
Response Time ⁶ τ (63%)		5		30	s
Long term drift			< 0.04		°C/yr

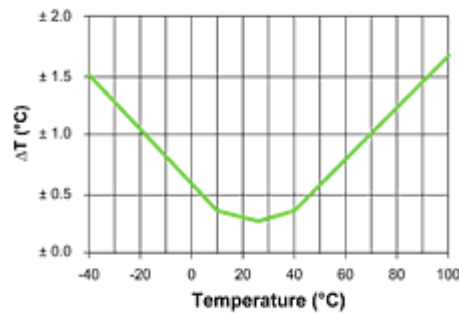


Fig. 4.5 Parámetros del sensor (temperatura) y curva de precisión

El esquema de conexión muestra como se conecta a un micro controlador (en nuestro caso usando la placa Arduino).

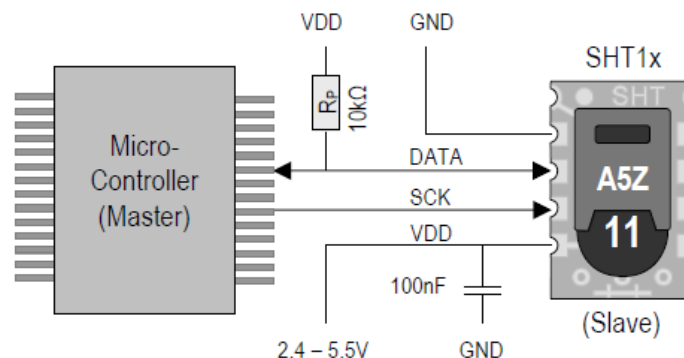


Fig. 4.6 Esquema de conexión

4.3.1.2. Funcionamiento del sensor

Como se muestra en la figura 4.6, se conectan únicamente cuatro cables, dos para alimentarlo (V_{cc} y Gnd) y dos para la comunicación (SCK y Data).

Para alimentar el sensor, se pueden usar tensiones desde 2.4V hasta 5.5V, pero usaremos 3.3V que es un valor que la placa Arduino nos ofrece. Los demás parámetros eléctricos los podemos observar en el datasheet anexo A6.1. Conectamos masa (Gnd) a la misma placa.

Para realizar la comunicación nos servimos de los pines SCK y Data, que proporcionan interfaz I^2C (Fig. 4.7). I^2C es un bus de comunicaciones en serie que utiliza dos líneas para transmitir la información: una para los datos y otra para la señal de reloj.

- SCK: (Serial Clock Input): Lo usamos para sincronizar la comunicación entre el micro controlador y el SHT15.
- DATA: Para enviar comandos al sensor, SCK debe estar en el flanco de subida y se envía información hasta que cambie de estado. Para leer datos del sensor válidos, leemos después del cambio de estado del SCK a nivel bajo y se mantiene hasta un cambio de flanco.

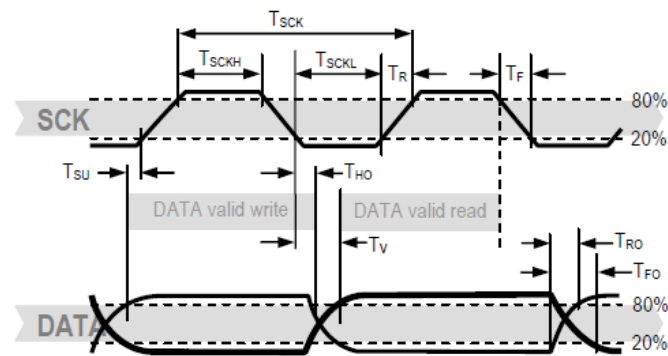


Fig. 4.7 Funcionamiento de I²C del sensor

4.3.1.3. Código

El código Arduino para leer la información del sensor SHT15 se muestra a continuación:

```
// SHT15 Sensor Coefficients
const float C1=-4.0; // for 12 Bit
const float C2= 0.0405; // for 12 Bit
const float C3=-0.0000028; // for 12 Bit
const float D1=-40.1; // for 14 Bit @ 5V
const float D2=0.01; // for 14 Bit DEGC
const float T1=0.01; // for 14 Bit @ 5V
const float T2=0.00008; // for 14 Bit @ 5V

void resetSHT()
{
  pinMode(dataPin,OUTPUT);
  pinMode(sckPin,OUTPUT);

  shiftOut(dataPin, sckPin, LSBFIRST, 255);
  shiftOut(dataPin, sckPin, LSBFIRST, 255);

  digitalWrite(dataPin,HIGH);
  for(int i = 0; i < 15; i++){
    digitalWrite(sckPin, LOW);
    digitalWrite(sckPin, HIGH);
  }
}
```

```

void startSHT()
{
    pinMode(sckPin,OUTPUT);
    pinMode(dataPin,OUTPUT);
    digitalWrite(dataPin,HIGH);
    digitalWrite(sckPin,HIGH);
    digitalWrite(dataPin,LOW);
    digitalWrite(sckPin,LOW);
    digitalWrite(sckPin,HIGH);
    digitalWrite(dataPin,HIGH);
    digitalWrite(sckPin,LOW);
}

void writeByteSHT(byte data)
{
    pinMode(sckPin,OUTPUT);
    pinMode(dataPin,OUTPUT);
    shiftOut(dataPin,sckPin,MSBFIRST,data);
    pinMode(dataPin,INPUT);

    //Wait for SHT15 to acknowledge by pulling line low
    while(digitalRead(dataPin) == 1);

    digitalWrite(sckPin,HIGH);
    digitalWrite(sckPin,LOW); //Falling edge of 9th clock

    //wait for SHT to release line
    while(digitalRead(dataPin) == 0 );

    //wait for SHT to pull data line low to signal measurement completion
    //This can take up to 210ms for 14 bit measurments
    int i = 0;
    while(digitalRead(dataPin) == 1 )
    {
        i++;
        if (i == 255) break;
        delay(10);
    }
    i *= 10;
}

int readByte16SHT()
{
    int cwt = 0;
    unsigned int bitmask = 32768;
    int temp;

    pinMode(dataPin,INPUT);
    pinMode(sckPin,OUTPUT);

    digitalWrite(sckPin,LOW);

```

```

for(int i = 0; i < 17; i++) {
    if(i != 8) {
        digitalWrite(sckPin,HIGH);
        temp = digitalRead(dataPin);
        cwt = cwt + bitmask * temp;
        digitalWrite(sckPin,LOW);
        bitmask=bitmask/2;
    }
    else {
        pinMode(dataPin,OUTPUT);
        digitalWrite(dataPin,LOW);
        digitalWrite(sckPin,HIGH);
        digitalWrite(sckPin,LOW);
        pinMode(dataPin,INPUT);
    }
}

//leave clock high??
digitalWrite(sckPin,HIGH);
return cwt;
}

int getTempSHT()
{
    startSHT();
    writeByteSHT(B00000011);
    return readByte16SHT();
}

int getHumidSHT()
{
    startSHT();
    writeByteSHT(B000000101);
    return readByte16SHT();
}

void loop () {
    delay(2000); //Every two seconds it will send information about sensors

    int temp_raw = getTempSHT(); // get raw temperature value
    BT.print("T");
    float temp_degc = (temp_raw * D2) + D1; // Unit Conversion - See datasheet
    serialPrintFloat(temp_degc);

    int rh_raw = getHumidSHT(); // get raw Humidity value
    BT.print("H");
    float rh_lin = C3 * rh_raw * rh_raw + C2 * rh_raw + C1; // Linear conversion
    float rh_true = (temp_degc * (T1 + T2 * rh_raw) + rh_lin); // Temperature compensated RH
    serialPrintFloat(rh_true);
}

```

4.3.2. Gas

4.3.2.1. Elección del sensor de CO₂

Se han valorado dos soluciones para implementar el sensor, mostradas en la siguiente tabla. Uno debe alimentarse y medir mediante una resistencia de carga (electroquímico) y el otro electromecánico devuelve una diferencia de potencial.

Tabla 4.2. Tabla de características de los sensores CO2-D1 y MG811

	CO2-D1		MG811
Sensitivity	mV/decade concentration change (0.5% to 5% CO ₂)	6 to 10	=====
Response time	t90 (s) for mV change (20°C)(0.5% to 5% CO ₂)	2-4 mins	20 sec
Zero	E0 @ 5000ppm CO ₂	-30 to +30mV	30-50mV
Resolution	RMS noise (ppm equivalent) @ 5,000ppm CO ₂	100	=====
Range	CO ₂ concentration	0.2% to 95%	350-10000ppm
Operating life	months until 80% original signal	24 month warranted	=====
Temperature range	=====	10 to 35° C	-20° C to 50° C
Pressure range	=====	80 to 120 kPa	=====
Humidity range	=====	15 to 95 % rh	=====
Input Impedance of op amp input	=====	>100MΩ	100—1000GΩ
Power Cons.	=====	=====	1200mW
Price	=====	31,25€	=====

Se escoge el sensor de Alphasense por su bajo consumo y porque no requiere alimentación, las características básicas de este son:

- Bajo coste.
- Tamaño reducido.
- No requiere alimentación.
- Detecta de 0,2% a 95% de concentración de CO₂.
- Opera de 10 a 35° C de temperatura.
- Opera de 15 a 95% humedad relativa.

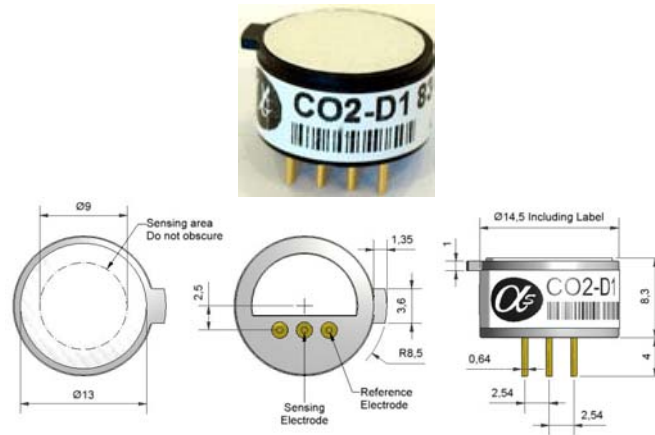


Fig. 4.8 Sensor CO2-D1, dimensiones y pines

Este sensor tiene dos electrodos para medir el valor, el electrodo de referencia y el electrodo de medida. La diferencia de potencial que observamos entre los dos pines nos devuelve un valor entre -30 y 30mV, esto hace que debamos amplificar la señal, para ello usamos un amplificador de instrumentación, detallado en el apartado 4.3.2.3.

La diferencia de potencia debe ser medida con un circuito con alta impedancia, para no dañar el sensor. Como se observa en la figura 4.9, la sensibilidad del sensor no es lineal, varía dependiendo de la concentración, la sensibilidad aumenta a mayor concentración.

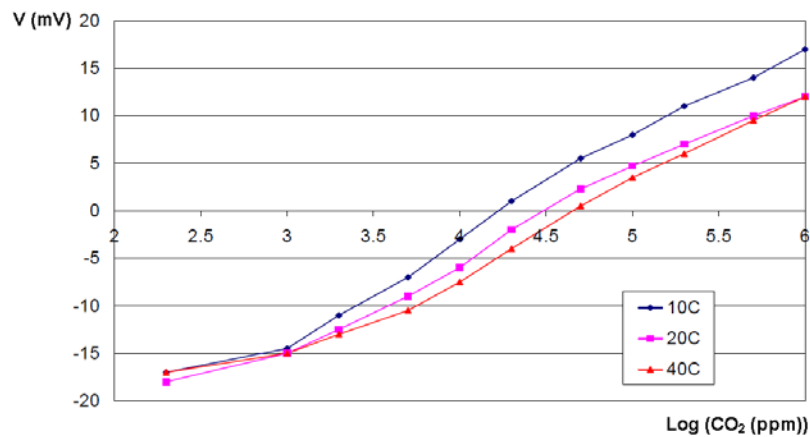


Fig. 4.9 Curva de respuesta del sensor

Con la ayuda de esta gráfica podemos determinar la concentración en ppm que obtenemos del sensor.

Para obtener la concentración de CO₂ basta con determinar en el valor del eje X según el valor que devuelve el sensor (eje Y) y aplicarle el logaritmo inverso.

Debemos realizar una aproximación para determinar en que punto de la recta nos encontramos, para ello obtenemos la ecuación de la recta para los tres rangos. Aproximamos linealmente los valores de la gráfica para poder obtener una ecuación.

La gráfica aproximada linealmente se muestra en la siguiente figura:

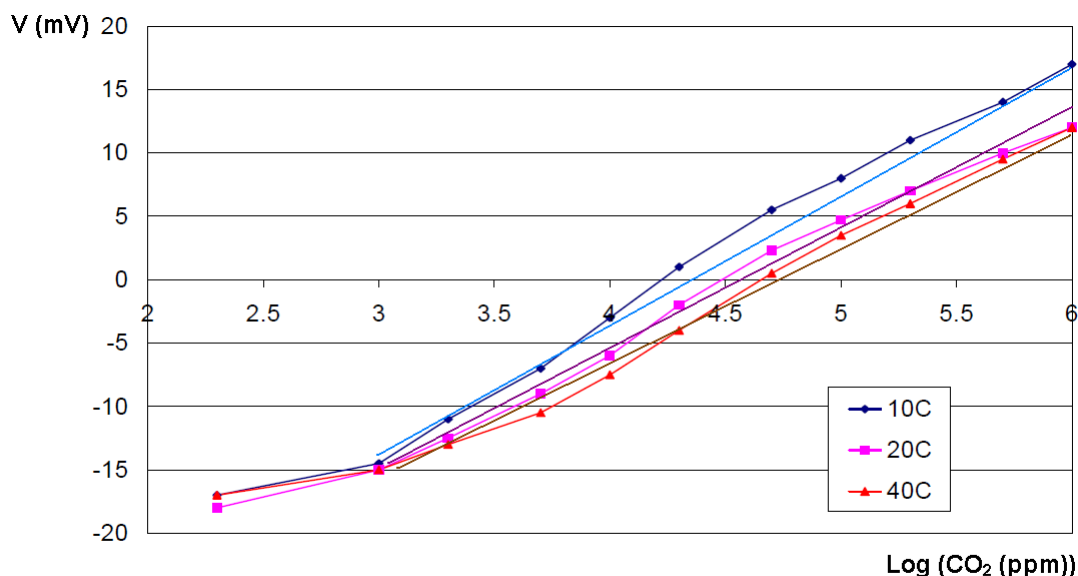


Fig. 4.10 Aproximación lineal a la gráfica de respuesta del sensor

De esta gráfica se obtienen los siguientes valores:

Tabla 4.3. Tabla usada para obtener las ecuaciones en función de la temperatura

Temperatura	$T < 15^\circ \text{C}$		$15^\circ \text{C} > T < 35^\circ \text{C}$		$T > 35^\circ \text{C}$	
	X	Y	X	Y	X	Y
Valores	4,3	0	4,55	0	4,7	0
	4,8	5	5,0625	5	5,2	5
Pendiente	$m = (y_2 - y_1) / (x_2 - x_1)$		$m = (y_2 - y_1) / (x_2 - x_1)$		$m = (y_2 - y_1) / (x_2 - x_1)$	
	10		9,756097561		10	
Ec. Recta	$x = ((y - y_1) / m) - x_1$		$x = ((y - y_1) / m) - x_1$		$x = ((y - y_1) / m) - x_1$	
	$(y - 5) / (10 - 4,8)$		$(y - 5) / (9,756097561 - 5,0625)$		$(y - 5) / (10 - 5,2)$	

A partir de estos valores (aproximados linealmente), sacamos la pendiente de las tres rectas mediante la ecuación:

$$m = \left(\frac{y_2 - y_1}{x_2 - x_1} \right) \quad (4.1)$$

Obtenemos:

$$m(A)=10, m(B)=9,7561 \text{ y } m(C)=10 \quad (4.2)$$

Ahora buscamos la ecuación de cada recta:

$$y - y_1 = m(x - x_1) \quad (4.3)$$

Y obtenemos las siguientes relaciones que nos permitirán obtener el valor del eje de las x ($\log(\text{CO}_2)$) sabiendo el valor devuelto por el sensor:

$$A: X = \frac{Y - 5}{10} - 4,8 \quad (4.4)$$

$$B: X = \frac{Y - 5}{9,7561} - 5,0625 \quad (4.5)$$

$$C: X = \frac{Y - 5}{10} - 5,2 \quad (4.6)$$

4.3.2.2. Adaptación de la señal del sensor para leer desde Arduino

Arduino lee o captura el valor de entrada de cualquier pin analógico, la placa Arduino realiza una conversión analógica a digital de 10 bits. Esto quiere decir que mapear los valores de voltaje de entrada, entre 0 y 5 voltios, a valores enteros comprendidos entre 0 y 1024.

La placa Arduino lee señales analógicas pero las convierte a digital con una precisión de $5/1024$, esto significa que podemos detectar cambios de hasta 48,828mV.

El inconveniente es que la señal debe ser positiva para poder leerla correctamente desde Arduino. Para ello necesitamos adaptar la señal que entrega el sensor.

4.3.2.3. Adaptación de la señal que entrega el sensor

Para poder leer bien la señal entregada por el sensor, nos servimos de un amplificador de instrumentación. Se han comparado dos amplificadores, el INA126 y el AD620 con las siguientes características principales:

Tabla 4.4. Tabla comparativa de amplificadores

	INA126	AD620
POWER SUPPLY (current per channel)	175 μ A/chan	1300 μ A/chan
SUPPLY RANGE	$\pm 1.35\text{V}$ to $\pm 18\text{V}$	$\pm 2.3\text{V}$ to $\pm 18\text{V}$
OFFSET VOLTAGE	250mV max	50 mV max
OFFSET DRIFT	3mV/ $^{\circ}$ C max	0.6 mV/ $^{\circ}$ C max
NOISE	35nV/ $\sqrt{\text{HZ}}$	9 nV/ $\sqrt{\text{HZ}}$ @ 1 kHz
INPUT BIAS CURRENT	25nA max	1nA max
TEMPERATURE RANGE	-40 $^{\circ}$ C to +85 $^{\circ}$ C	-40 $^{\circ}$ C to +85 $^{\circ}$ C
GAIN	1 to 10000	1 to 1000

Ambos amplificadores permiten ajustar la ganancia con una sola resistencia externa:

DESIRED GAIN (V/V)	R_G (Ω)	NEAREST 1% R_G VALUE	1% Std Table Value of R_G , Ω	Calculated Gain	0.1% Std Table Value of R_G , Ω	Calculated Gain
5	NC	NC				
10	16k	15.8k	49.9 k	1.990	49.3 k	2.002
20	5333	5360	12.4 k	4.984	12.4 k	4.984
50	1779	1780	5.49 k	9.998	5.49 k	9.998
100	842	845	2.61 k	19.93	2.61 k	19.93
200	410	412	1.00 k	50.40	1.01 k	49.91
500	162	162	499	100.0	499	100.0
1000	80.4	80.6				
2000	40.1	40.2	249	199.4	249	199.4
5000	16.0	15.8	100	495.0	98.8	501.0
10000	8.0	7.87	49.9	991.0	49.3	1,003

Fig. 4.11 Ganancia según resistencia en los amplificadores

Hemos escogido el INA126 pese a que los márgenes de error sean peores, pero su consumo es realmente bajo como se observa en la tabla 4.4 y nos permite una ganancia bastante más elevada que el AD.

La relación de ganancia viene determinada por esta ecuación:

$$G = 5 + \frac{80K\Omega}{R_G} \quad (4.7)$$

Para obtener una ganancia de 20 aproximadamente se ha escogido una resistencia de 4,7K Ω (valor comercial):

$$G = 5 + \frac{80K\Omega}{4,7K\Omega} = 22,021 \text{ (Este es nuestro factor de ganancia V/V teórico)} \quad (4.8)$$

El esquema usado es el siguiente:

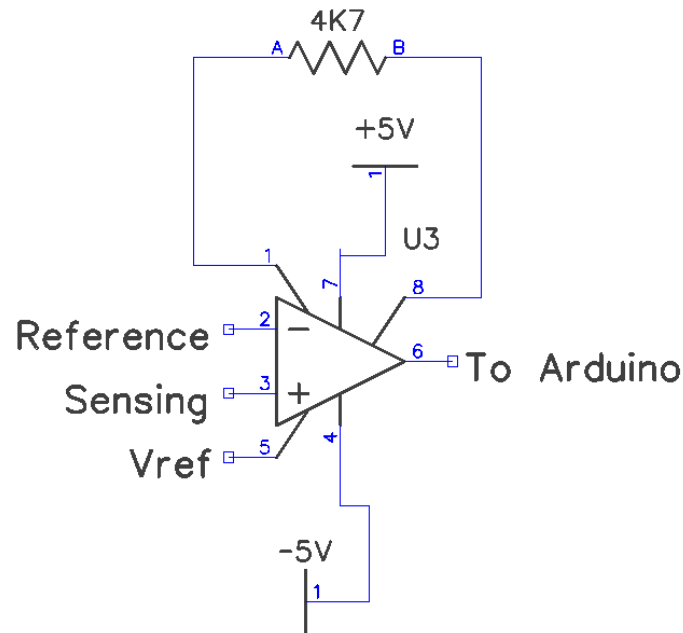


Fig. 4.12 Esquema de conexionado del amplificador

Para desplazar la señal hacia valores positivos únicamente, debemos usar V_{ref} , inicialmente con un potenciómetro para hacer las pruebas pertinentes y posteriormente añadiendo un divisor de tensión para lograr el desplazamiento deseado.

Los valores absolutos devueltos por el sensor son -30mV a +30mV, aplicándole la ganancia la salida absoluta del amplificador de instrumentación es:

$$V_{out \max}_{Amplificador} = V_{in \max}_{sensor} * G = \pm 30mV * 22,021 = \pm 660,64mV \quad (4.9)$$

Nos fijamos en los valores negativos, se debe desplazar el máximo negativo para que se sitúe en el margen positivo. Para ello bastaría con desplazarla 660, 64mV, de esta forma se asegura que la salida del amplificador siempre sea positiva. Se decide aplicar un offset de aproximadamente 1V, aplicamos el divisor de tensión con el siguiente esquema donde V_{in} la tomamos de 5V y V_{out} a 1V.

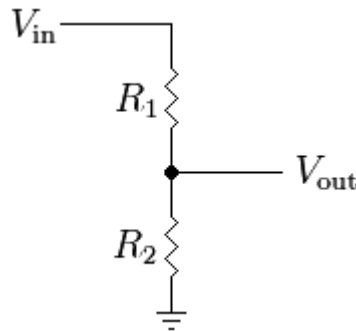


Fig. 4.13 Divisor de tensión

Calculamos mediante la fórmula del divisor:

$$V_{out} = \frac{R_2}{R_1 + R_2} \cdot V_{in} \quad (4.10)$$

Fijamos R_2 a 10KΩ;

$$\begin{aligned} V_{out}(R_1 + R_2) &= R_2 \cdot V_{in} \\ \frac{V_{out}}{V_{in}}(R_1 + R_2) &= R_2 \\ R_1 &= \frac{R_2 - \frac{V_{out}}{V_{in}} \cdot R_2}{\frac{V_{out}}{V_{in}}} = 40K\Omega \end{aligned} \quad (4.11)$$

El valor comercial más próximo a R_1 es 39KΩ, por lo tanto $V_{offset}(V_{out})$ será de 1,02V.

El valor de V_{ref} que aplicamos al amplificador también lo mandamos a la placa Arduino para que lo convierta a digital (entre 0 y 1024). Esto se hace para que el valor de V_{ref} sea siempre el real y no el estimado. Cuando el sistema se queda sin batería o encontramos una fluctuación no deseada en el voltaje, el valor de V_{ref} puede variar, de este modo aseguramos estabilidad al sistema con una lectura del valor entregado fiable. El esquema resultante es el siguiente:

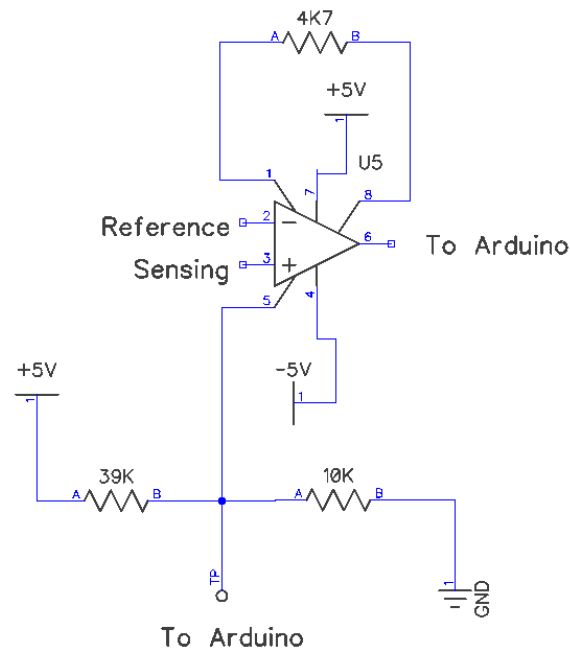


Fig. 4.14 Esquema usado de conexionado

4.3.2.4. Fuente de alimentación simétrica

El amplificador necesita alimentación simétrica, nuestra placa Arduino sólo entrega voltaje positivo (3,3V o 5V), por eso debemos implementar una fuente siguiendo este esquema:

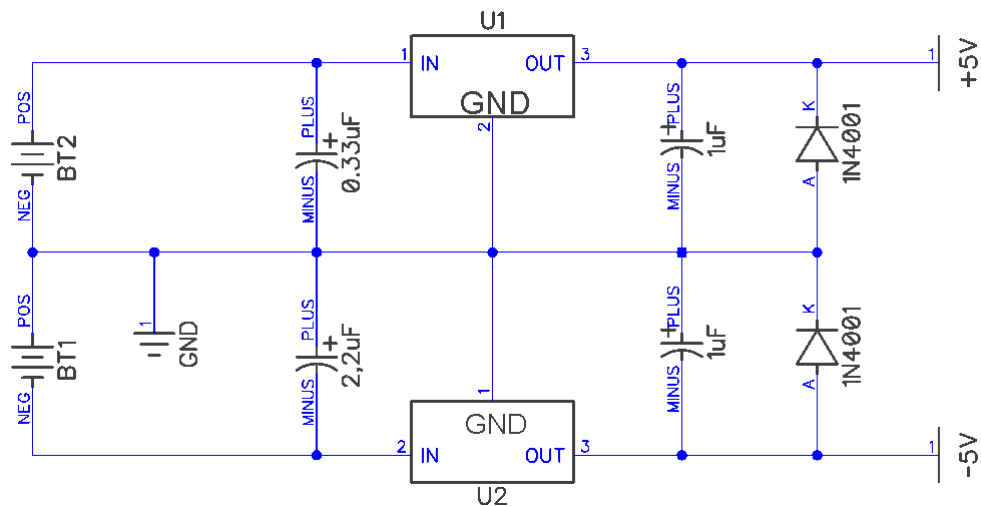


Fig. 4.15 Esquema de la fuente de alimentación

Se ha usado +/-9V usando dos pilas de 9V conectadas de la siguiente forma:

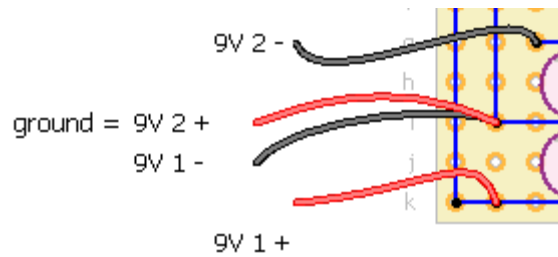


Fig. 4.16 Conexión de las dos pilas

Con esta fuente se obtiene la tensión negativa para poder amplificar la señal correctamente.

4.3.2.5. Recuperar la señal que envía Arduino

Para poder recuperar la señal y determinar la concentración de CO₂ se deben seguir los siguientes pasos:

1. Se envían dos valores:

Valor que entrega el sensor (amplificado y desplazado).
Ejemplo: Valor gas = 67.

Valor Vref (el valor que indica cuanto se ha desplazado).
Ejemplo: Valor offset = 204.

Estos valores han sido digitalizados y están comprendidos entre 1 y 1024, debemos determinar el valor correspondiente analógico:

$$Valorgas(analógico / amplificado / offset) = Valorgas \cdot \frac{5}{1024} = 0,327(V) \quad (4.12)$$

$$ValorOffset(analógico) = ValorOffset \cdot \frac{5}{1024} = 0,996(V) \quad (4.13)$$

2. Aplicarle el Offset: Restarle ValorOffset (analógico) a ValorGas (analógico/amplificado/offset).

$$\begin{aligned} Valorgas(analógico / amplificado) &= \\ &= Valorgas(analógico / amplificado / offset) - ValorOffset(analógico) = \quad (4.14) \\ &= 0,327V - 0,996V = -0,669V \end{aligned}$$

3. Restarle la ganancia a ValorGas (analógico/amplificado). Recordemos que $G=22,021$.

$$Valorgas(analógico) = \frac{Valorgas(analógico / amplificado)}{G} = \frac{-0,669V}{22,021} = -30,38mV \quad (4.15)$$

Este valor corresponde al eje de las Y(mV) de la gráfica de la figura 4.10.

4. Aplicar las ecuaciones según la temperatura obtenida por el sensor de temperatura SHT15 para determinar el valor de X según la gráfica:

Estamos a 24° C:

$$X = \frac{Y - 5}{9,7561} + 5,0625 = \frac{-30,38 - 5}{9,7561} + 5,0625 = 1,436 \quad (4.16)$$

5. Obtener la concentración de CO₂ (en ppm).

$$\begin{aligned} \log(CO_2) &= 1,436 \\ CO_2(ppm) &= 10^{1,436} = 27,29 ppm \end{aligned} \quad (4.17)$$

4.3.2.6. Evaluar la concentración de CO₂ y sus síntomas

El sensor devuelve un valor en ppm, podemos mostrarlo como porcentaje de concentración en el aire (en tanto por ciento). 1 ppm es $1/1000000 = 0.000001$ que corresponde a 0.0001%.

La concentración de dióxido de carbono al aire libre oscila entre 360 ppm en áreas de aire limpio y 700 ppm en las ciudades. El valor máximo recomendado para los interiores es de 1.000 ppm.

El dióxido de carbono sólo es perjudicial a partir de una concentración de un 5% del volumen (que son 50.000 ppm), no obstante a partir de concentraciones mucho menores (a partir de valores entre 800 y 2.000 ppm) se pueden producir molestias diversas, como dolor de cabeza, cansancio, pérdidas de concentración y bajo rendimiento. En la siguiente figura se muestran las afectaciones al organismo dependiendo de la concentración en el aire:

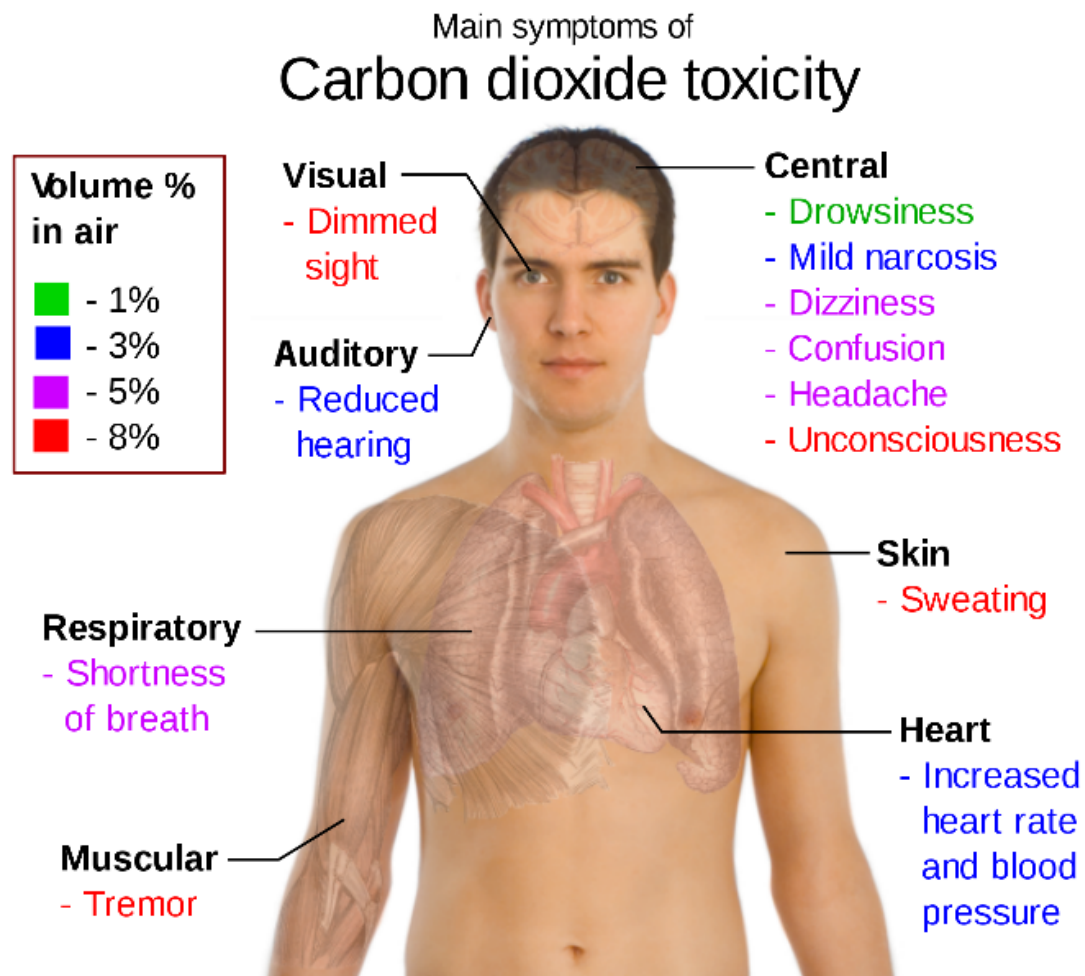


Fig. 4.17 Afectaciones al organismo según concentración de CO₂

Se obtendrá una alarma cuando la concentración en aire sea de 4% o superior (40.000 ppm) que advertirá de la peligrosidad del entorno.

4.3.3. Otros gases

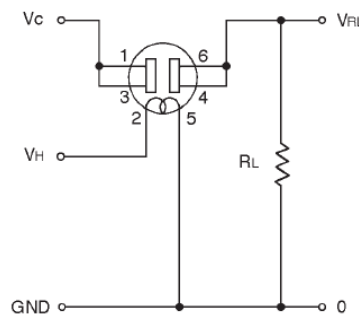
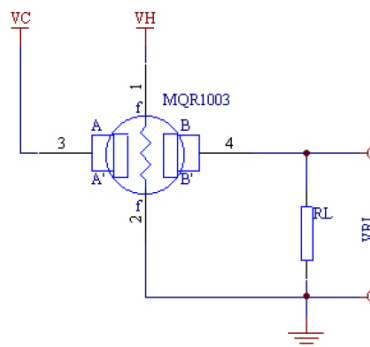
Se pretende utilizar otro tipo de sensores como hidrógeno, gas licuado (LPG), metano y gas inflamable.

Se encontraron dos posibles opciones para detectar gas hidrógeno, las alternativas eran, sensor MQR-1003-H y Fígaro TGS821.

Tabla 4.5. Tabla comparativa de sensores de hidrógeno

		Fígaro TGS821	MQR-1003-H
Heater Voltage	V_h	$5 \pm 0,2V$ (AC or DC)	$5 \pm 2\%(DC)V$
Circuit Voltage	V_c	24V max DC	15V max DC
Load Resistance	R_l	0,45K Ω variable	variable
Sensor Resistance	R_s	At 100 ppm 1K Ω -10 K Ω	=====
Change ratio of Sensor Resistance	R_s/R_0	0,6-1,2	=====
Heater Resistance	R_h	$38 \pm 3\Omega$ (room temperature)	$30 \pm 3\Omega$ (room temperature)
Heater Power Consumption	P_h	660mW (typ) at $V_h=5V$	833mW (typ)

Ambos se miden mediante una resistencia de carga (R_L). Figuras 4.18 y 4.19

**Fig. 4.18** Esquema para testear Figaro TGS821**Fig. 4.19** Esquema para testear MQR1003

Los valores de V_c y V_h son similares en los dos casos, la Resistencia interna del sensor varia en ambos sensores dependiendo de R_L

El datasheet del sensor Fígaro nos indica que antes de su correcta operación pasan 7 días, por este motivo se escoge el sensor MQR (con un tiempo de respuesta de 30s aproximadamente), que a su vez lo encontramos en forma de pack, incluyendo los sensores requeridos.

Después de disponer de los sensores e intentar realizar algún test con ellos, se observa la poca claridad del datasheet, de sus gráficas y la información difusa, por este motivo y por el tiempo que pasaría hasta recibir uno nuevo y después para adaptarlo; se hace imposible adaptar estos sensores. Se adjunta datasheet de ambos sensores en el anexo A6.2.

4.3.4. Seguridad

Se puede dar la circunstancia que el bombero se quede quieto en una posición varios minutos, el receptor GPS envía la posición y todo parece correcto. ¿Que pasa si el bombero se ha quedado inconsciente en el suelo?

Para solucionar este problema se ha introducido un sensor tilt muy sensible que devuelve en cada momento un indicador de si el bombero esta en movimiento o no, cuando no este en movimiento después de 12 segundos se enviará una alarma.

El sensor tilt funciona como un interruptor, alimentándolo a 5V y con una resistencia a masa como se muestra en la figura 4.20.

Cuando se cierra el circuito (tilt activo), Arduino lee los 5V de la entrada digital.

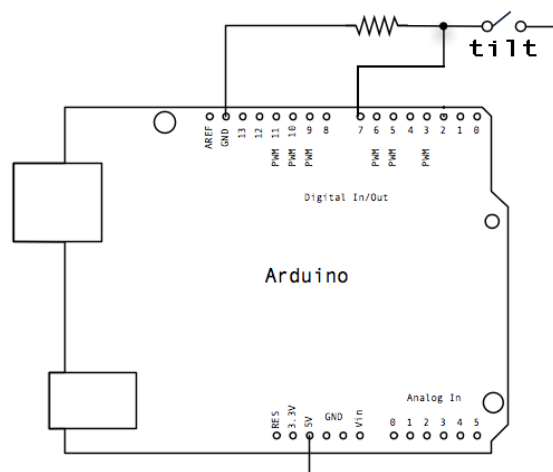


Fig. 4.20 Esquema de conexionado del sensor de seguridad

4.4. Conectividad Bluetooth

4.4.1. Introducción

Para conseguir conectividad entre la placa Arduino y la PDA se ha escogido un módulo Bluetooth embedded, adaptable a cualquier dispositivo y permite una fácil configuración.

Bluetooth es una especificación industrial para Redes Inalámbricas de Área Personal (WPAN's) que permite la transmisión de voz y datos entre diferentes dispositivos mediante radiofrecuencia usando la banda ISM de 2,4GHz. Facilita la comunicación entre dispositivos fijos y móviles.

4.4.2. Elección del módulo

Se han comparado dos módulos BT en la siguiente tabla comparativa entre uno del fabricante "Firmtech" y otro de "Roving Networks" (usado en alguna versión de Arduino BT):

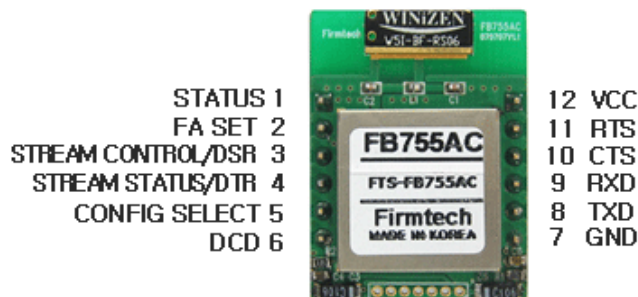
Tabla 4.6. Tabla comparativa de dispositivos Bluetooth

	FB755AX	RN-41
Frequency in Use	2.400 - 2.4835 GHz	2402 - 2480MHz
Reliability	FHSS (Frequency Hopping Spread Spectrum) 79 channels	FHSS/GFSK modulation, 79 channels at 1MHz intervals
Transmission Rate	1Mbps ~ 3Mbps	1.5Mbps sustained, 3.0Mbps burst in HCI mode
Transmission Output	1mW(10m,Class2), 100mW(100m Class1)	=====
Network Configuration	Allow simultaneous connections up to 7 devices (in case of ACL)	=====
Bluetooth Specification	2.0 Support	2.1/2.0/1.2/1.1 Support
Communication distance	100 m	100 m
Frequency Range	2.4 GHz ISM Band	2.4 GHz ISM Band
Transmit Power	16dBm (Typical)	12dBm
Sensitivity	-83dBm (Typical)	-80dBm (Typical)
Support Bluetooth Profile	GAP, SPP	GAP, SDP
Input Power	3.3V	3.3V

Current Consumption	100 mA (Max)	30mA connected, <10mA sniff Mode
Operating Temperature	-10° C – 70° C	-40° C - 85° C
Communication Speed	1,200bps – 230,400bps	1200bps up to 921Kbps, non-standard baud rates can be programmed.
Interface	UART (TTL Level)	UART (SPP or HCI) and USB (HCI only)
Flow Control	RTS, CTS, DTR, DSR support	RTS, CTS, DTR, DSR support
	Class 1 + EDR	v2.0+EDR support
Price	26,05€	59,95\$

Los dos son bastante similares, pero se ha elegido el de Firmtech por los siguientes motivos:

1. Soporta comandos AT y funciona a nivel TTL (adaptable a la placa Arduino).
2. La configuración por defecto se adapta a nuestras necesidades.
3. Permite crear Bluetooth Piconets en el caso de ser necesario.
4. Es más económico y mantiene las prestaciones.



Dimension : 20.5 mm(Width) * 27.7 mm (Length) *12 mm (Height)

Fig. 4.21 Configuración de los pines del módulo Bluetooth

4.4.3. Configuración básica BT: comunicación con Arduino

Para establecer la comunicación serie entre BT y Arduino se ha usado la librería “NewSoftSerial” que permite superar la limitación de la placa “Arduino Duemilanove” en cuanto a conexiones serie. Arduino proporciona una comunicación serie, que es usada para conectar mediante USB al PC y cargar el programa, mediante la librería nueva, se pueden asignar dos pines para que realicen una nueva conexión, simplemente añadiendo:

```
#include <NewSoftSerial.h> //Connecting BT device

NewSoftSerial BT (5, 6);
```

A partir de la asignación de los dos pines con el nombre de BT, se substituye el comando Serial de nuestra Arduino por BT, teniendo las mismas funciones: print, println...

Por defecto el módulo viene con una configuración que nos sirve para nuestras necesidades:

TYPE	SET VALUE
Device Name	FB755v.x.x.x
Pin Code(Pass key)	BTWIN
Uart(baud rate-data bit-parity bit-stop bit)	9600-8-N-1
ROLE	SLAVE
Connection Mode	MODE4 (AT command)
Operation Mode	MODE0 (1:1 communication)
Debug char	0x02

Fig. 4.22 Configuración por defecto según el fabricante

Es necesario inicializar desde la placa Arduino el módulo, para ello se usa el siguiente trozo de código extraído del apéndice proporcionado por el fabricante para configurar como esclavo (después debemos buscar y conectar desde PC o PDA para iniciar la comunicación):

HOST → BT : **AT**␣

BT → HOST : **OK**␣(normally execute the command language)

HOST → BT : **AT+BTSCAN**␣ (SCAN command language)

BT → HOST : **OK**␣ (normally executes the command language)

BT → HOST : **ERROR**␣ (does not execute the command language properly)

Fig. 4.23 Comunicación BT-HOST según anexo del fabricante

Código implementado para establecer conexión:

```
void setup() {
  pinMode(dataPin, OUTPUT);
  pinMode(sckPin, OUTPUT);
  pinMode(vibroPin, INPUT);

  //BT start config
  BT.begin(9600);
  BT.print(2, BYTE);
  BT.print("AT");
  BT.print(13, BYTE);
  delay(1000); //Necessary for sending inialitation code OK
  BT.print("AT+BTSCAN");
  BT.print(13, BYTE);

  resetSHT();

  //Timer Start
  MsTimer2::set(1000, tick2); // 1000ms period
  MsTimer2::start();
}
```

El código envía primero el carácter de debugar: 0x02, y seguidamente envía un comando AT y “enter” (correspondiente al carácter 13 en la tabla Ascll) para establecer comunicación entre Arduino y BT, añadimos un delay de un segundo ya que si no ha respondido el módulo BT y le enviamos el siguiente comando, nos devolverá un error. Finalmente enviamos comando AT+BTSCAN y “enter”. A partir de este momento el módulo será detectable desde cualquier dispositivo Bluetooth.

4.5. Consumo

El consumo se ha calculado con un amperímetro en serie con las dos pilas del circuito. El consumo de la fuente positiva es de 58,7mA, mientras que el de la fuente negativa es de 1,36mA, esto indica que se consumirá antes una que otra. Una solución sería usar una batería recargable.



Fig. 4.24 Cálculo de amperaje consumido por el circuito

Para obtener el número de horas que podemos usar el dispositivo, se requiere conocer cual es la carga de las pilas. Según la gráfica del datasheet mostrada en la figura 4.25 para carga continua obtenemos aproximadamente 500mAh.

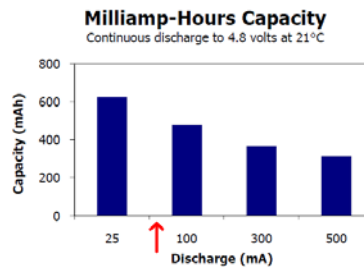


Fig. 4.25 Gráfica de carga de la pila

Por lo tanto la vida de las pilas de nuestro circuito es de:

$$N^{\circ} horas = \frac{500mAh}{60,06mA} = 8,32horas \quad (4.18)$$

4.5.1. Solución a los problemas de consumo

Si la batería dura 8,32 horas es probable que se agote cuando se está realizando una labor de extinción. Para corregir esto se deja la opción a realizar modificaciones futuras que lo mejoren:

- Uso de una batería recargable en primer lugar.
- En segundo lugar programar Arduino con interrupciones para que se encienda 20 segundos cada minuto o algo similar, de esta manera la duración de la pila es notablemente más elevada. Para realizar esto se debe tener en cuenta que si apagamos el BT, perdemos conexión con la PDA, pero existen comandos AT para gestionar el módulo BT que permiten una reconexión con el dispositivo conectado anteriormente, o también se puede conectar siempre con el mismo dispositivo.

4.6. Esquema y montaje

A continuación se muestra una imagen del prototipo (Fig. 4.26) y el circuito esquemático completo (Fig. 4.27). En el anexo 2 se incluyen más fotografías del montaje.

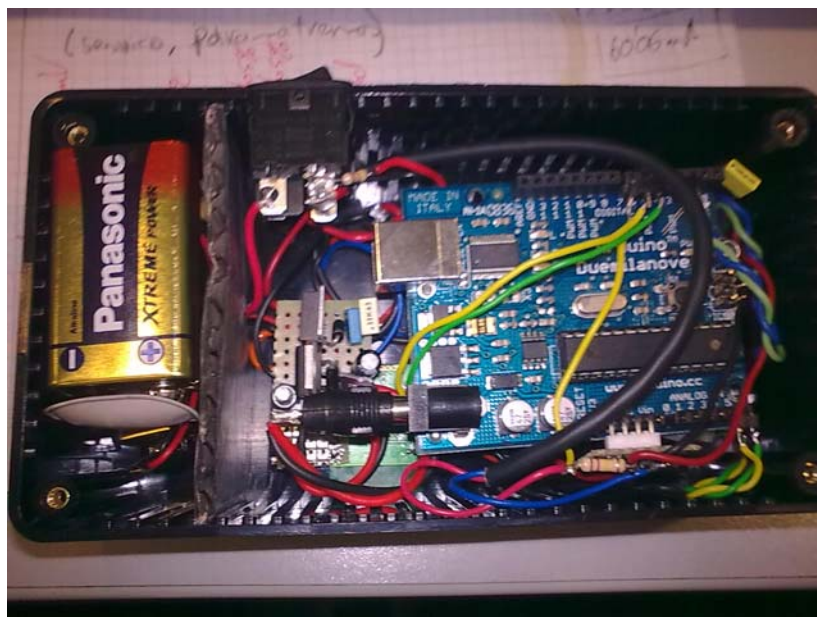


Fig. 4.26 Circuito definitivo con Arduino y pilas



Fig. 4.27 Montaje definitivo del módulo del bombero

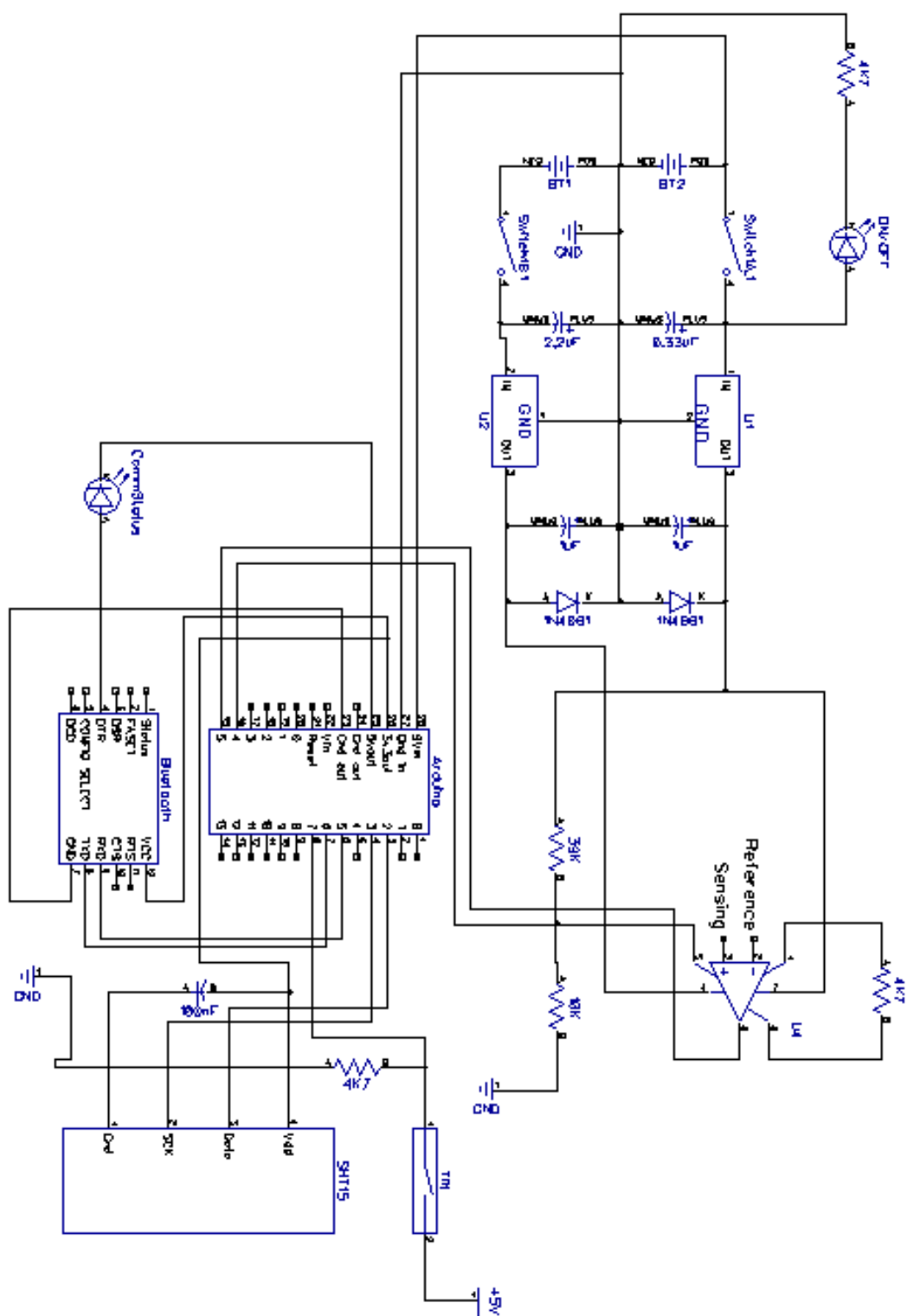


Figura 4.28 Esquemático del circuito

CAPÍTULO 5: MIGRACIÓN DEL MIDDLEWARE MAREA AL ENTORNO .NET COMPACT FRAMEWORK

5.1. Introducción

En la primera parte de este capítulo se presenta el middleware MAREA implementado por el grupo ICARUS de la escuela politécnica superior de Castelledefels.

En la segunda parte se introducen los cambios realizados en este para su compatibilidad con cualquier dispositivo móvil que sea compatible con la plataforma *.NET Compact Framework*.

5.2. MAREA

Un middleware se puede definir como una capa de software que ofrece una conectividad y servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas.

El sistema descrito está desarrollado sobre el middleware MAREA (Middleware Architecture for Remote Embedded Applications). Tiene una arquitectura Publish/Suscribe que se basa en el paso de mensajes entre un elemento que actúa como “publicador” de la información y un elemento que actúa como “suscriptor”. Cuando existen nuevos datos el publicador envía esa nueva información a todos los suscriptores que estaban registrados (ver [2]).

MAREA es capaz de gestionar y controlar esta información y mensajes de tal manera que sea totalmente transparente para cada uno de los servicios. Esta transparencia permite comunicar diferentes servicios sin necesidad de saber la ubicación de los mismos.

MAREA nos ofrece cuatro primitivas de servicio para realizar la comunicación. Son variables, eventos, funciones y ficheros:

- **Variables:** Se utilizan para enviar información normalmente periódica, como ejemplo en este proyecto se utiliza para enviar la información de la posición y medidas de diferentes sensores.
- **Eventos:** Son una primitiva de servicio muy parecida a las variables, pero se puede asegurar que la comunicación será fiable. Se utilizan más en información con cierta importancia y que normalmente no es periódica. Este sistema utiliza estas primitivas con el sensor de movimiento (Tilt).
- **Funciones:** Estas primitivas requieren una comunicación bidireccional ya que se realiza una petición de la información que le será entregada por

el servicio requerido. En este sistema se utilizarán para hacer peticiones a una base de datos orientada a objetos.

- **Ficheros:** Se utilizan en caso de que la información que se publica sea de gran tamaño. Esta información viaja como *stream*, de tal manera que puede obtenerse y guardarse en un fichero.

La utilización de las primitivas de servicio depende de la información y de la comunicación requerida. En este sistema se han usado tres tipos de primitivas (variables, eventos y funciones) para aprovechar todas sus ventajas y demostrar también la eficiencia de este middleware. En la figura se presenta un esquema del sistema completo con las principales primitivas que se utilizan para comunicar los diferentes servicios a través de MAREA.

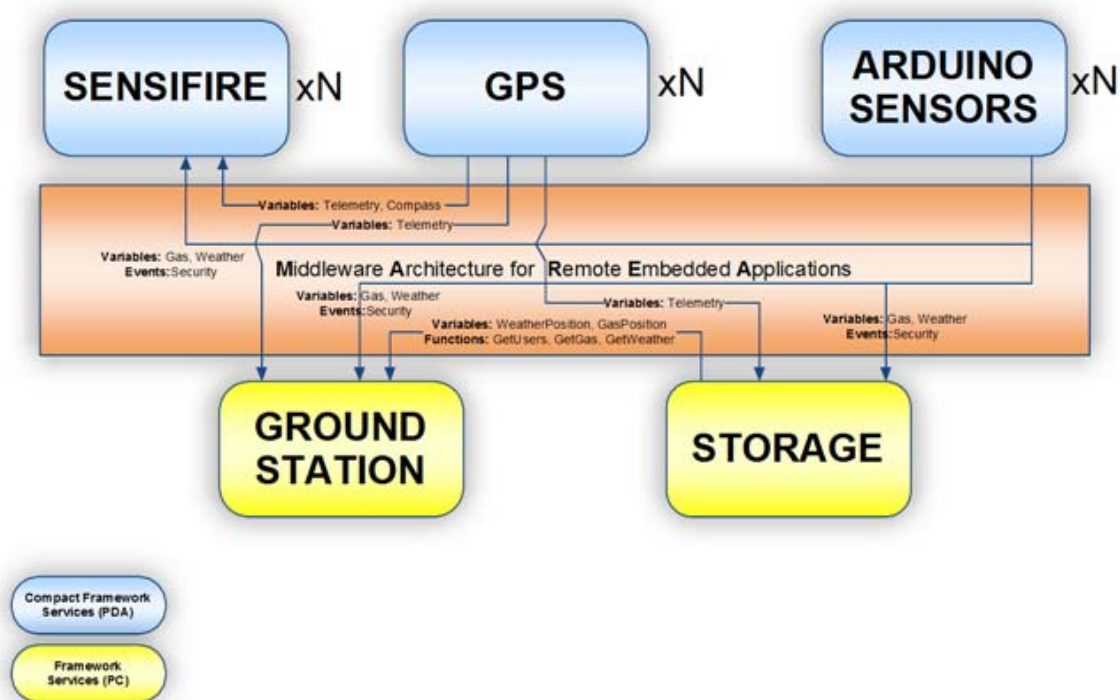


Fig. 5.1 Primitivas de servicio entre los diferentes servicios de MAREA

Uno de los principales beneficios que se pueden obtener de este middleware es la API (Application Programming Interface) de programación que dispone. Es fácil y sencilla de utilizar. Cada servicio dispone de un contenedor y unas funciones predeterminadas. Entre las funciones disponibles podemos destacar las siguientes:

- **Start:** Esta función gestiona el arranque del servicio, se registra el servicio en el middleware y se indica a que variables o eventos está suscrito el servicio.

- **Stop:** Esta función indica la parada del servicio. Se invierte el proceso de registro que implica automáticamente también dejar de estar suscrito a las variables y eventos que tenía registrados.
- **ServiceDescription:** En ella se describe el servicio y las variables, eventos y funciones que tiene para publicar. Es útil para comunicar al middleware las primitivas de servicios de las cuales va ser el publicador.
- **VariableChanged:** Cuando el publicador envíe una nueva variable a la que el suscriptor este registrado se llamará a esta función. Esta función recibe dos parámetros de entrada, el identificador de la variable y el valor de la variable. Por tanto la función *VariableChanged*, es la que recibirá y procesará la nueva variable.
- **EventFired:** Esta función la podemos describir de manera análoga a la anterior, pero en vez de recibir una variable, estaremos trabajando con eventos.
- **FunctionCall:** Cuando nos llegue una petición de una primitiva de función invocará este método. Los parámetros de entrada que tiene es el nombre de la función que solicitan y los parámetros que requiere dicha función. Cuando se ejecute la función solicitada devolverá el valor oportuno.

La manera de publicar una variable, evento o función se realiza con las mismas funciones explicadas antes (*VariableChanged*, *EventFired* y *FunctionCall*), pero invocándolas desde el objeto que contiene el contenedor del servicio y con los parámetros de los objetos que vayamos a publicar. Para publicar un fichero se utiliza el método *PublishFile* para publicarlo en MAREA y se le envía un evento al suscriptor con el nombre del fichero para que lo recoja invocando al método *GetFile* e informándole del nombre.

Gracias a este middleware podemos reemplazar la utilidad de un servicio por otro que ofrezca las mismas características sin verse afectado el correcto funcionamiento del sistema y podemos aprovechar esta ventaja para trabajar con componentes reales o simulados indistintamente.

5.3. Migración al entorno .NET Compact Framework

5.3.1. Introducción

En el siguiente apartado se presentan principales cambios añadidos al middleware MAREA para su compatibilidad con dispositivos móviles (PDA's y Smartphones). La adaptación de MAREA a .NET Compact Framework ha sido necesaria para la posterior implementación del proyecto.

A partir de este momento se utilizara la nomenclatura "MAREACF" para referirse a la versión *.NET Compact Framework* de dicho middleware.

5.3.2. Estructura del código y archivos del proyecto

En el siguiente apartado se hace una breve explicación de la reestructuración del código y ficheros realizada para la implementación de MAREA CF.

En vistas a reducir el código y cantidad de archivos de que utiliza este middleware, la versión Framework y Compact Framework utilizan los mismos archivos de código fuente. La existencia de código fuente en archivos diferentes para cada uno de los proyectos supondría por una parte un aumento del tamaño total del programa y por otra la repetición de código común para las dos versiones (MAREA y MAREA CF).

Además la utilización de un mismo archivo de código fuente facilita mucho las futuras modificaciones, ya que los cambios en código compartido por los dos proyectos solo se tienen que hacer una vez.

La utilización de los mismos archivos de código para las dos proyectos obliga a la utilización de la directivas `#if` para que cuando se compile una de las dos soluciones el compilador de Visual Studio pueda diferenciar entre que código debe compilar.

A continuación se muestra un ejemplo de la estructura de las directivas `#if`:

```
#if (PocketPC)
//Todo el codigo dentro de esta sentencia se compilaría en
MAREA
#else
//Todo el codigo dentro de esta sentencia se compilaría en
MAREACF
```

Aunque las dos versiones utilicen el mismo archivo de código fuente, los archivos de proyecto (*.csproj) cambian ya que son diferentes para soluciones MAREA y MAREA CF.

En los siguientes pasos se explica cómo generar los archivos de proyectos y solución para que apunten al mismo código.

1. Se genera un proyecto vacío (MareaCF.csproj) de SmartDevice con su solución (MareaCF.sln).

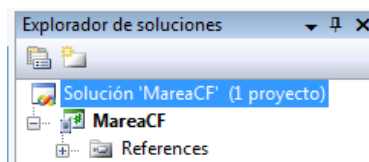


Fig. 5.2 Creación del proyecto MareaCF

2. Se copia el proyecto (MareaCF.csproj) en la carpeta donde se encuentra el mismo archivo de la versión de Framework (Marea.csproj).
3. Se copia la solución (MareaCF.sln) en la carpeta donde se encuentra el mismo archivo de la versión Framework (Marea.sln).
4. Se abre la solución de Compact Framework (MareaCF.sln) con un editor de texto y se modifica el directorio principal en el que se guarda, la misma. Con esto se pretende que las dos soluciones (*.sln) estén en una misma carpeta de manera que no se tenga que tener una carpeta para cada una de los dos soluciones. Por ejemplo, en la siguiente línea "MareaCF\MareaCF.csproj" se tendría que substituir por "Marea\MareaCF.csproj".

```
Project("{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}") = "MareaCF",  
  "MareaCF\MareaCF.csproj", "{D71EDC71-7BA3-4E48-8B8F-  
  E60E45FA7B83}"  
EndProject
```

5. Llegados a este punto ya se dispone de una solución y un proyecto vacío en Compact Framework, al que se debe añadir el código fuente, carpetas y referencias de los archivos ya existentes de la versión Framework.

5.3.3. Consola

Uno de los problemas iniciales que surgieron en la implementación de MAREACF fue que este requiere la utilización de una consola en modo texto para poder ejecutarse (activar y desactivar servicios, ver información de: variables, eventos, funciones, ficheros...).

Esto no supone ningún problema en el entorno de *Framework*, ya que este dispone de una consola (System.Console) con la que es usuario puede interactuar. Por contra Compact Framework no dispone de esta. Por lo tanto el diseño de una consola se hace imprescindible.

La consola básicamente diseñada consiste en un formulario con dos TextBox's, uno para leer y otro para escribir, y un botón (Enter) para entrar el texto. Se ha intentado asemejar al máximo su funcionamiento y opciones al de la consola de *.NET Framework* (System.Console).

La consola diseñada nos permite leer cadenas de texto y escribir: enteros, cadenas de texto, números de coma flotante, decimales, parámetros...

Si se quiere utilizar una misma consola en diferentes clases a la vez, debemos llamar a una instancia del objeto de tipo *CompactConsole* en cada una de las diferentes clases en que se vaya a utilizar.

```
CompactConsole Compact = CompactConsole.GetInstance();
```

Para ejecutar la consola debemos crear un thread para iniciarla:

```
Thread th = new Thread(new  
ThreadStart(Compact.Console.Start));  
th.IsBackground = true;  
th.Start();
```

Para cerrarla debemos ejecutar el siguiente código:

```
Compact.Console.Stop();
```

En la siguiente figura se muestra el aspecto de la consola de MAREACF.



Fig. 5.3 Consola implementada para la versión Compact Framework de MAREA

5.3.4. Librerías

Aunque la mayoría de librerías existentes para Framework tienen sus equivalentes en Compact Framework, hay algunas excepciones que provocan que se tengan que buscar soluciones.

La finalidad de este proyecto no es hacer una versión compatible en Compact Framework del middleware Marea, sino que esta es una parte necesaria para el desarrollo del mismo. Por este motivo inicialmente la primera solución planteada fue encontrar librerías similares, de código abierto, intentado reducir al máximo el tiempo de elaboración de esta tarea.

.NET Compact Framework es una plataforma que tiene más de 7 años de vida, además durante estos la aparición en el mercado de nuevos dispositivos móviles (PDA's y smartphones) ha impulsado notablemente su uso. Por lo que encontrar librerías alternativas no ha sido un problema.

A continuación se muestra una tabla de las equivalencias entre librerías existentes en .NET Framework y las librerías utilizadas no existentes para .NET Compact Framework que se utilizan para llevar a cabo las mismas tareas.

Tabla 5.1. Librerías utilizadas para la migración de MAREA al entorno Compact Framework

Clase/Librería .NET Framework	Descripción	Alternativa utilizada	Alternativas testeadas no funcionales
Semaphore (System.Threading)	Clase especial protegida para restringir o permitir el acceso a recursos compartidos en un entorno de multiprocesamiento	NativeSync (Joel Ivory Johnson)	.NET Compact Framework compatibility
(System.Net. NetworkInformation)	Espacio de nombres que proporciona acceso a los datos del tráfico de red, a la información de la dirección de red y a la notificación de cambios de dirección para el equipo local.	OpenNETCF.Net	
BinaryFormatter (System.Runtime. Formatters.Binary)	Serializa y deserializa objetos en formato binario	AsGoodAsItGets. Serialization, MareaCoder	CFormatter 1.0 Beta, CFormatter+

5.3.5. Cargador de servicios

Una biblioteca de enlace dinámico o dll (dynamic-link library) es el término con el que se refiere a los archivos con código ejecutable que se cargan bajo demanda de un programa por parte del sistema operativo. Esta denominación es exclusiva a los sistemas operativos Windows siendo “dll” la extensión con la que se identifican estos ficheros.

El middleware MAREA se ha diseñado para cargar dinámicamente los diferentes servicios a partir de las dll's. La versión Framework carga los servicios automáticamente a partir de un archivo xml (startup.xml) en que se especifica el nombre del servicio. Mediante el método `AppDomain.GetAssemblies()` se obtiene una lista de las dll agregadas como referencia al proyecto Marea.csproj y las clases que contiene la misma.

Posteriormente se comprueba si la clase que contiene el servicio que se desea cargar existe en el contexto de ejecución del dominio de MAREA. En caso que así sea se obtiene información de su dll mediante el método `Type.GetType()` y se carga el servicio.

.NET Compact framework no dispone del método `AppDomain.GetAssemblies()`, por lo tanto no podemos saber las dll que han sido agregadas. Hay que encontrar alguna manera de obtener información de las dll que contienen los servicios que se desean cargar.

Para cargar los servicios en MAREA utiliza un string con el siguiente formato (variable `strNameTemp`):

```
Type t = Type.GetType(strNameTemp);  
if (t != null)  
{  
    return Activator.CreateInstance(t);  
}
```

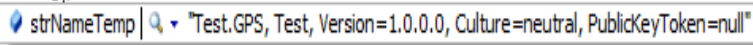


Fig. 5.4 Cadena de caracteres utilizada para cargar un servicio

Después de algunas pruebas se llegó a la conclusión de que conociendo los dos primeros campos que contienen el espacio de nombres, clase y nombre de la dll era suficiente para cargar los servicios de MAREA.

Así que lo que se hizo es crear un archivo xml (startuppcf.xml) que además de contener el nombre del servicio (clase), añadiera el espacio de nombres de este. De este modo no es necesaria la utilización del método `AppDomain.GetAssemblies()` para cargar servicios. Es decir no es necesario buscar la información de la dll, sino que esta se obtiene del archivo startuppcf.xml.

5.3.6. Traza y control de mensajes

MAREA utiliza la librería *Log4Net* para guardar trazas de todo lo que hace. La utilización de esta librería es especialmente útil en aplicaciones que se encuentran en desarrollo, ya que gracias a esta se pueden encontrar posibles errores que surjan durante la ejecución.

Una de las funcionalidades de *Log4Net* es redirigir la traza de los mensajes a diferentes destinos (fichero de texto, consola).

Debido a que MAREACF posee una consola propietaria, se debe especificar de alguna manera que estos mensajes se muestren por esta. Para esto creamos la siguiente clase ("appender" de *Log4Net*). Lo que hace es llamar a una instancia de la consola de MAREACF y devolver todo los mensajes que le lleguen.

```
public class log4netcfConsole_Appender : AppenderSkeleton
{
    protected override void Append(LoggingEvent
loggingEvent)
    {
        Comapct_Console Compact =
        Compact_Console.GetInstance();

        Compact.Console.WriteLine(RenderLoggingEvent(log
gingEvent));
    }
}
```

Para que se muestren los mensajes según el "appender" anterior debemos añadir en el archivo de configuración de log4net de MAREACF (Appcf.config) lo siguiente:

```
<appender name="Console_Appender" type="log4cf_Appender">

    <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%date [%thread] %-
5level %logger - %message%newline" />
    </layout>

</appender>
```

Una de las ideas para desarrollar la aplicación es que la red sea autónoma y no requiera de dispositivos de interconexión adicionales para su funcionamiento (por ejemplo un acces point). Para conseguir esto nos servimos de MCL (software desarrollado por Microsoft para crear redes mesh), sobre una red wireless que implementa tecnología ad-hoc.

Los bomberos estarán conectados entre sí cuando estén en el radio de alcance, este radio lo determina cada PDA de cada bombero, este puede quedarse momentáneamente fuera de la red, pero cuando se acerque otra vez a la red, automáticamente se conectará, son auto ruteables. La red demuestra ser muy confiable.

Las redes mesh son de topología en malla donde cada nodo está conectado a todos los nodos, de esta manera es posible enviar mensajes entre nodos por diferentes caminos. Cada nodo realiza las funciones de enrutado, siendo de esta manera mucho más eficiente que si un dispositivo tuviese que encaminar todos los paquetes de todos los nodos.

Algunas ventajas de las redes mesh:

- Es posible llevar los mensajes de un nodo a otro por diferentes caminos.
- No puede existir absolutamente ninguna interrupción en las comunicaciones.
- Cada servidor tiene sus propias comunicaciones con todos los demás servidores.
- Si falla un cable el otro se hará cargo del tráfico.
- No requiere un nodo o servidor central lo que reduce el mantenimiento.
- Si un nodo desaparece o falla no afecta en absoluto a los demás nodos.

CAPÍTULO 6: IMPLEMENTACIÓN DISTRIBUIDA EN SERVICIOS

6.1. Introducción

El sistema se ha implementado en una arquitectura basada en servicios (SOA) sobre el middleware MAREA. Tal como se muestra en la figura 7.1 existen cinco servicios principales:

- GPS
- Arduino Sensors
- Sensifire
- Storage
- Ground Station

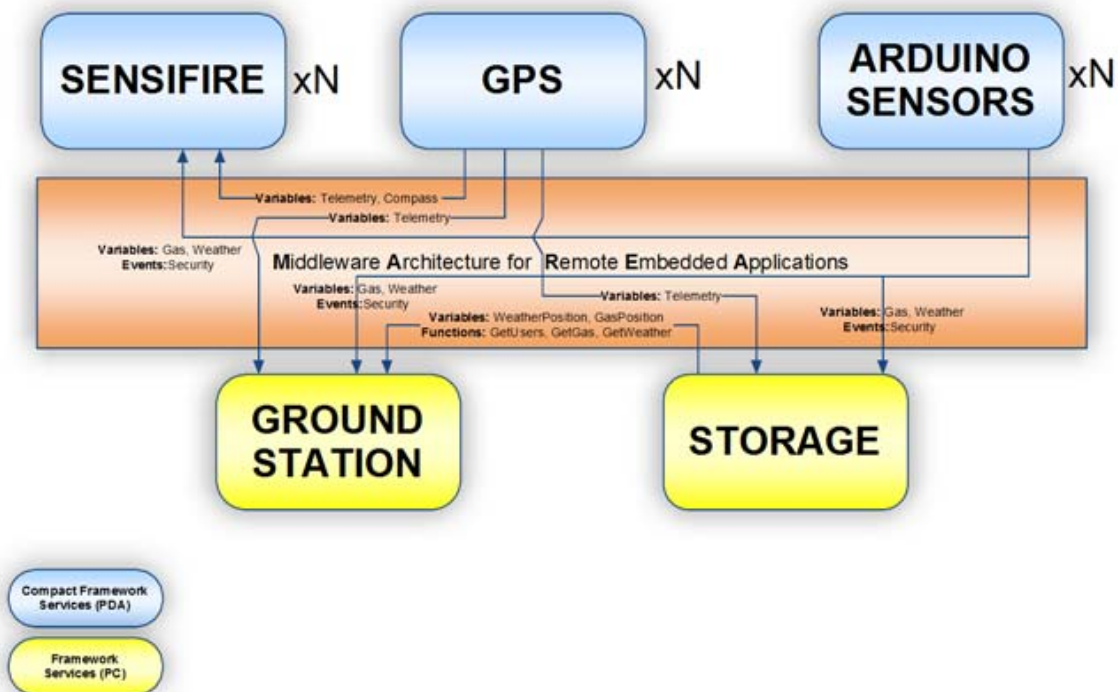


Fig. 6.1 Primitivas de servicio entre los diferentes servicios de MAREA

Como se trata de un sistema distribuido se puede garantizar robustez. Además cada uno de los componentes que proporcionan información a estos servicios se pueden reemplazar por cualquier otro que ofrezca la misma información.

En el siguiente capítulo se explicarán la implementación de estos cinco servicios. Los servicios *GPS*, *Arduino Sensors* y *Sensifire* han sido diseñados

para utilizarse en la distribución *Compact Framework* de MAREA. Es decir, se han implementado para que funcionen en un dispositivo móvil (PDA). Por otro lado el servicio *Storage y Ground Station* lo hacen en la distribución *Framework* de MAREA (PC).

El middleware MAREA es el encargado de comunicar estos servicios aunque se ejecuten en maquinas distintas o presenten arquitecturas diferentes (PC-PDA).

6.2. Servicio: GPS

El servicio *GPS* es el encargado de leer e interpretar las sentencias NMEA que llegan a través del receptor GPS de una PDA.

Para su implementación se ha utilizado la librería *WindowsMobile.Samples.Location* de Microsoft.

Para iniciar y parar el servicio en MAREA utilizamos el método *Open* y *Close* de esta librería.

```
//Start GPS service  
gps.Open();  
//Stop GPS service  
gps.Close();
```

Una de las particularidades de esta librería es la autogestión de la conexión con el dispositivo GPS. La librería *WindowsMobile* busca el puerto en el que se encuentra el dispositivo GPS y configura la conexión con el dispositivo de modo transparente para el usuario. Esto se hace particularmente útil para este proyecto ya que en el escenario planteado se utilizan múltiples PDA's que no tienen por qué tener el dispositivo GPS asociado al mismo puerto serie.

Mediante un manejador de eventos *LocationChangedEventHandler* propio de *WindowsMobile* capturamos la latitud, longitud, altitud, rumbo (ver [8]) y *una referencia de tiempo* cuando la señal del dispositivo GPS es válida.

```

gps.LocationChanged += new LocationChangedEventHandler
(gps_LocationChanged);

protected void gps_LocationChanged(object sender,
DeviceStateChangedEventArgs args)
{
    if (gps.Opened)
    {
        if (device_position != null)
        {
            if (device_position.LatitudeValid)

                Telemetry.Latitude =
                    device_position.Latitude;

            if (device_position.LongitudeValid)

                Telemetry.Longitude =
                    device_position.Longitude;

            if (device_position.SeaLevelAltitudeValid)

                Telemetry.Altitude =
                    device_position.SeaLevelAltitude;

            if (device_position.HeadingValid)

                Compass = device_position.Heading;
        }
    }
}

```

Posteriormente el servicio pública estas variables en una clase *Telemetry* del tipo *Position*. Esta será consumida por los siguientes servicios: *Sensifire*, *Ground Station* y *Storage*.

Además de la variable *Telemetry* este servicio pública la variable *Compass* que indica el rumbo que está siguiendo el bombero. Esta variable únicamente será consumida por el servicio *Sensifire* del mismo bombero que la pública.

Para seguir un estándar en todos los servicios, las variables de posición se transmiten en formato decimal.

La variable *Telemetry* se publica con una cadencia de 30 segundos, por otra parte *Compass* lo hace cada 5 segundos.

6.3. Servicio: Arduino Sensors

El servicio *Arduino Sensors* se encarga de leer e interpretar las medidas de los sensores que se reciben a través del dispositivo Bluetooth conectado a la placa *Arduino Duemilanove*.

Para el correcto funcionamiento de este servicio es necesario previamente asociar el dispositivo Bluetooth con la PDA.

Para leer los datos transmitidos por el dispositivo Bluetooth es necesario configurar y abrir el puerto serie que le ha sido asignado a dicho dispositivo. Esto se hace en el método *Start* del servicio de MAREA.

```
public void Open()  
{  
    try  
    {  
        Port.BaudRate = 9600;  
        Port.PortName = "COM7";  
        Port.DtrEnable = true;  
        Port.RtsEnable = true;  
        Port.DataBits = 8;  
        Port.StopBits = StopBits.One;  
        Port.Parity = Parity.None;  
        Port.Handshake = Handshake.None;  
        Port.ReadTimeout = 30000;  
        Port.WriteTimeout = 30000;  
        Port.Open();  
        Port.DataReceived += new  
            SerialDataReceivedEventHandler(port_DataReceived);  
        weather.State = ArduinoSate.Connected;  
    }  
    catch (Exception ex)  
    {  
        weather.State = ArduinoSate.NotConnected;  
  
        Compact.Console.WriteLine("\r\nUnable to connect  
with the Arduino shiled: "+ex.Message);  
    }  
}
```

La comunicación con el dispositivo Bluetooth es unidireccional, únicamente debemos leer del puerto serie. Para ello se utiliza el manejador de eventos *SerialDataReceivedEventHandler*, de manera que cada vez que se reciben datos nuevos por el puerto serie, estos se procesan para identificar las medidas que contienen.

```
private void port_DataReceived(object sender,
SerialDataReceivedEventArgs e)
{
    string Sentence;
    Sentence = Port.ReadLine();
    Value = Sentence.Substring(1);
    switch (Sentence.Substring(0, 1))
    {
        case "T":
            weather.Temperature = Convert.ToDouble(Value);
            break;
        case "H":
            weather.Humidity = Convert.ToDouble(Value);
            break;
        ...
    }
}
```

Este servicio publica dos variables (*Weather*, *Gas*) y un evento (*Security*):

- Weather: Es una clase que contiene los valores de humedad y temperatura procedentes del sensor *SHT15*. Además incluye una referencia temporal y un identificador.
- Security: Es una clase que contiene un identificador, referencia temporal y un variable que nos indica si el sensor tilt detecta o no movimiento.
- Gas: Es una clase que contiene los valores de los diferentes sensores de gas (Metano, Hidrogeno, Dióxido de Carbono, gas licuado y gas inflamable).Además incluye una referencia temporal y un identificador.

Estas tres variables son consumidas por los siguientes servicios: *Sensifire*, *GroundStation* y *Storage*.

Para finalizar el servicio debemos cerrar el puerto serie correctamente para que cuando lo volvamos a utilizar este disponible.

6.4. Servicio: Sensifire

Este servicio contiene la interfaz que permite al bombero visualizar las medidas de los diferentes sensores, su posición y su rumbo. Para ello el servicio consume las variables *Weather*, *Gas*, *Telemetry* y *Compass* y el evento *Security*

La interfaz gráfica está formada por dos formularios:

- **Formulario principal:** Este formulario contiene los valores de las diferentes medidas tomadas por los sensores y la posición del bombero.

Las medidas se encuentran agrupadas según su tipo. En la parte superior se muestran las medidas ambientales, en el centro la posición del bombero y en la parte inferior las medidas de los diferentes sensores de gas.

Cada una de las medidas dispone de una barra que muestra el estado de la conexión con el dispositivo correspondiente. Las medidas ambientales y de gas provienen de la placa de sensores que se comunica con la PDA mediante el dispositivo Bluetooth. En caso que el puerto serie asociado a dicha conexión este abierto correctamente, la barra de estado aparecerá de color verde. En caso contrario se mostrará de color rojo.

En el caso de la posición, la barra muestra tres estados diferentes:

- Rojo: Dispositivo GPS no encontrado o error al abrir el puerto serie.
- Naranja: Dispositivo encontrado, señal GPS no válida
- Verde: Dispositivo encontrado, señal GPS válida

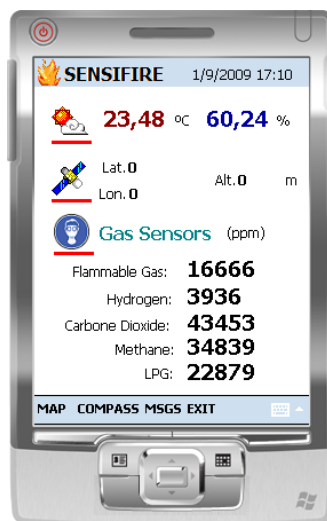


Fig. 6.2 Formulario principal del servicio Sensifire

- **Brújula:** Este formulario contiene una rosa de los vientos en que se muestra el rumbo que sigue el bombero. Esta es especialmente útil para orientar al bombero en situaciones en las que desconoce el terreno, como por ejemplo los incendios forestales.

La variable *Compass* (radianes) que publica el servicio *GPS*, permite ir actualizando el rumbo del bombero.

Este formulario utiliza dos PictureBox: uno que representa la rosa de los vientos. Y otro que se va moviendo, que se utiliza como puntero para mostrar el rumbo que sigue el bombero.

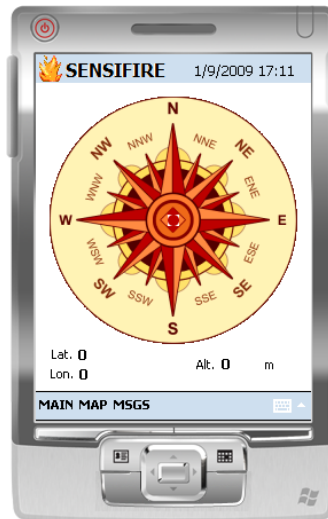


Fig. 6.3 Brújula implementada en el servicio *Sensifire*

Para dibujar correctamente el puntero sobre la rosa de los vientos se utilizan las siguientes expresiones que establecen la distancia, en pixeles, en la que se tiene que dibujar el puntero respecto al borde vertical izquierdo y horizontal derecho del formulario:

```
compass.Left = (int)(116 + 100*Math.Sin(f*Math.PI/180));
compass.Top = (int)(141 - 100 *Math.Cos(f*Math.PI/180));
```

6.5. Servicio: Storage

El servicio *Storage* se encarga de gestionar el funcionamiento de la base de datos *Perst*. Este servicio consume las variables *Gas*, *Weather*, *Telemetry* y el evento *Security* de los diferentes bomberos.

Además contiene las cinco funciones siguientes publicadas en MAREA para que el servicio *GroundStation* pueda realizar peticiones a la base de datos con el fin de recuperar el histórico de un bombero.

- *GetUsers*: Esta función devuelve una lista de todos los bomberos que estan o han estado registrados en el sistema.

- *GetGasByTime, GetWeatherByTime*: Las siguientes funciones devuelven las medidas de gas o medioambientales de un bombero concreto entre dos intervalos de tiempo.
- *GetGasByPosition, GetWeatherByPosition*: Estas funciones devuelven las medidas de gas o medioambientales de un bombero dado un rectángulo de coordenadas. Es decir retornan las medidas respectivas de un bombero que han sido tomadas dentro de una zona delimitada.

Cuando se llama alguna de las cuatro que devuelven medidas, la base de datos retorna objetos del siguiente tipo: *GasPosition, WeatherPosition*. Estos objetos además de contener la medida correspondiente contienen una referencia de donde fue tomada.

Al iniciarse y cerrarse este servicio la base de datos se abre y se cierra de forma automática.

6.6. Servicio: Ground Station

6.6.1. Introducción

El servicio *GroundStation* es el componente que contiene la interfaz gráfica que permite la monitorización del escuadrón terrestre de bomberos.

Al iniciar el servicio se abre un formulario que contiene la interfaz gráfica. Cuando este es creado se le pasa el objeto *service* y *servicecontainer* del servicio *GroundStation* para que pueda actuar como tal y pueda interactuar directamente con las variables del middleware.

Consecuentemente este servicio está compuesto por dos elementos importantes: *GroundStationService* (servicio) y *GroundStation* (formulario).

Por otra parte este servicio también es el encargado de realizar las peticiones WMS para la obtención de mapas.

6.6.2. Conexión WMS

Este servicio dispone de un segundo formulario que permite al usuario escoger el servidor del cual desea descargarse los mapas, mediante peticiones WMS.

Lo primero que deberá hacer el usuario nada más abrir este servicio es clicar el botón que se encuentra de la zona *WMS Connection* (zona 1 de la figura 6.8) para elegir una configuración y descargarse el mapa del servidor correspondiente.

A parte de esto podrá cambiar, si lo cree conveniente, las coordenadas por defecto en las que se debe centrar el mapa y las capas WMS que desee añadir.

Fig. 6.4 Formulario para la selección de servidor de mapas

Inicialmente el sistema se ha diseñado con la idea de trabajar con un único servidor de mapas propio. En vista a aumentar tolerancia a fallos se implemento esta funcionalidad. De este modo, en caso de que surja alguna incidencia con el servidor de mapas propio, el operador podrá descargarse los mapas desde cualquier otro servidor WMS disponible en Internet.

Las diferentes configuraciones de cada uno de los servidores se obtienen del archivo XML *wms_servers*. Cada una de las entradas contiene un identificador del servidor, la url para hacer peticiones WMS, el sistema de referencia espacial, las coordenadas por defecto y las capas disponibles.

```
<Server
name="ICC(Catalunya)"
url="http://shagrat.icc.es/lizardtech/iserv/ows?"
SpatialReferenceSystem="EPSG:4326" default_x="2.10"
default_y="41.35" default_zoom="0.03" format="jpeg">

    <layer>mtc5m</layer>
    <layer>orto5m</layer>

</Server>
```

6.6.3. Representación de datos

Para la representación de los bomberos en el mapa se utilizan iconos que muestran el tipo de medidas que se están visualizando (gas o ambientales). Además para diferenciar a cada uno de los bomberos se incluye una cadena de texto en la parte inferior del icono que muestra el identificador de bombero.

En la siguiente imagen se presentan los iconos utilizados para la representación de medidas ambientales y de gas respectivamente:



Fig. 6.5 Iconos utilizados en la interfaz gráfica del servicio

Para cambiar el tipo de medidas que se desean mostrar el usuario dispone de la zona *Layers* (zona 2 de la figura 6.8.). Para su implementación se utilizan capas vectoriales y de etiquetas de la librería *SharpMap* (*LabelLayer*, *VectorLayer*)

La consulta de las medidas de cada uno de los bomberos es interactiva, de manera que el usuario encargado de la monitorización puede consultarlas desplazando el puntero del mouse sobre la muestra deseada.



Fig. 6.6 Visualización de las medidas ambientales y de gas

Sharmap no contempla la utilización de este tipo de funcionalidades, por este motivo ha sido necesario programarlas.

Cada vez que el usuario mueve el mouse por encima de mapa se captura su posición en píxeles. Esta se compara con cada una de las posiciones de las diferentes muestras de los bomberos. Para ello es necesario recorrer el objeto del tipo *FeatureDataTable* de *SharpMap* que las contiene. También se utiliza el método *WorldToImage* para convertir las coordenadas en píxeles.

```
foreach (FeatureDataRow r in dtweather)
{
    PointF position_pixel =
    sharpMap.WorldToImage((Point)r.Geometry);

    if(mouse_position_pixel.X > position_pixel.X - 6 &&
    mouse_position_pixel.X < position_pixel.X + 6 &&
    mouse_position_pixel.Y > position_pixel.Y - 6 &&
    mouse_position_pixel.Y < position_pixel.Y + 6)
    {
        W_Form.Location = new
        Point(Convert.ToInt32(position_pixel.X) +
        mapImageRE.Location.X - 18,
        Convert.ToInt32(position_pixel.Y) +
        mapImageRE.Location.Y + 39);

        W_Form.Show_Weather(r["Id"].ToString(), Convert.ToDateTime(r["Time"]), Convert.ToDouble(r["Temperature"]), Convert.ToDouble(r["Humidity"]));

        W_Form.Show();
        showing_info = true;
    }
}
```

En caso de que el puntero se encuentre encima se muestra un formulario con las medidas correspondientes. Aquellos valores que estén fuera del umbral y puedan considerarse peligrosos para la integridad del bombero se muestran de color rojo.

6.6.4. Sistema de Alarmas

Con el fin de facilitar la tarea de monitorización, se ha incorporado un sistema de alarmas en tiempo real que permite controlar en todo momento las medidas de los sensores que se encuentren fuera de su respectivo umbral.

Estas se muestran y se ocultan automáticamente en la parte inferior derecha del mapa. Cada una de ellas contiene la siguiente información: fecha en la que fue tomada la medida, identificador del bombero, tipo de alarma (de gas o ambiental), la medida y valor de esta.



Fig. 6.7 Visualización de alarma de CO₂

Siempre que llegue una medida a través del middleware que este fuera del umbral, el sistema mostrará una alarma.

Comentar que el sistema de alarmas implementado sólo funciona en el modo Tiempo Real.

6.6.5. Modos de funcionamiento

Este servicio posee dos modos de funcionamiento. El usuario puede cambiar entre ellos mediante los botones de selección de la región *Mode* (zona 3 de la figura 6.8.).

- Tiempo Real: Permite visualizar las últimas medidas de gas o ambientales tomadas por cada uno de los bomberos. Cuando este modo esta activado, el servicio *GroundStationService* consume las variables *Telemetry*, *Gas*, *Weather* y *Security* que son publicadas por el servicio *Sensifire* y *ArduinoSensors*. Cada vez que llega un muestra nueva de un bombero esta es actualizada respecto a la anterior.
- Almacenamiento: Este modo permite hacer consultas a la base de datos para recuperar las medidas y el recorrido que ha hecho un bombero durante un periodo de tiempo concreto. Para poder hacer las consultas el formulario dispone de la región *Queries* (zona 4 de la figura 6.8.). Uno de los requisitos para poder trabajar en este modo es que el servicio *Storage*, que contiene la base de datos, este arrancado.

Para realizar la consulta se debe especificar el bombero sobre el cual se quiere hacer la misma. Esto se hace clicando sobre el desplegable *Name*. Al desplegarse el menú, se hace una llamada a la función *GetUsers* del servicio *Storage* que retorna la lista de bomberos que existen en la base de datos. Posteriormente se deben rellenar los intervalos de tiempo entre los cuales se desea hacer la búsqueda.

Finalmente con el botón *Do Query* se hace la petición llamando a la función correspondiente.

Para mostrar el resultado de la consulta se dibujan todas las muestras del bombero siempre que al hacerlo no se superpongan entre ellas. Para implementar esta funcionalidad, denominada proceso de *clustering*, se ha creado específicamente una clase en que se hace la siguiente comprobación: cada vez que se va a dibujar una nueva medida como resultado a la consulta, se buscan sus límites según el icono utilizado (en píxeles). En caso de que alguna de las muestras ya dibujadas se encuentre dentro de estos límites esta nueva medida no se dibuja. En caso contrario esta se dibuja.

También se incluyen líneas para unir cada una de las muestras con su anterior para poder facilitar la visualización de la ruta que ha seguido el bombero.

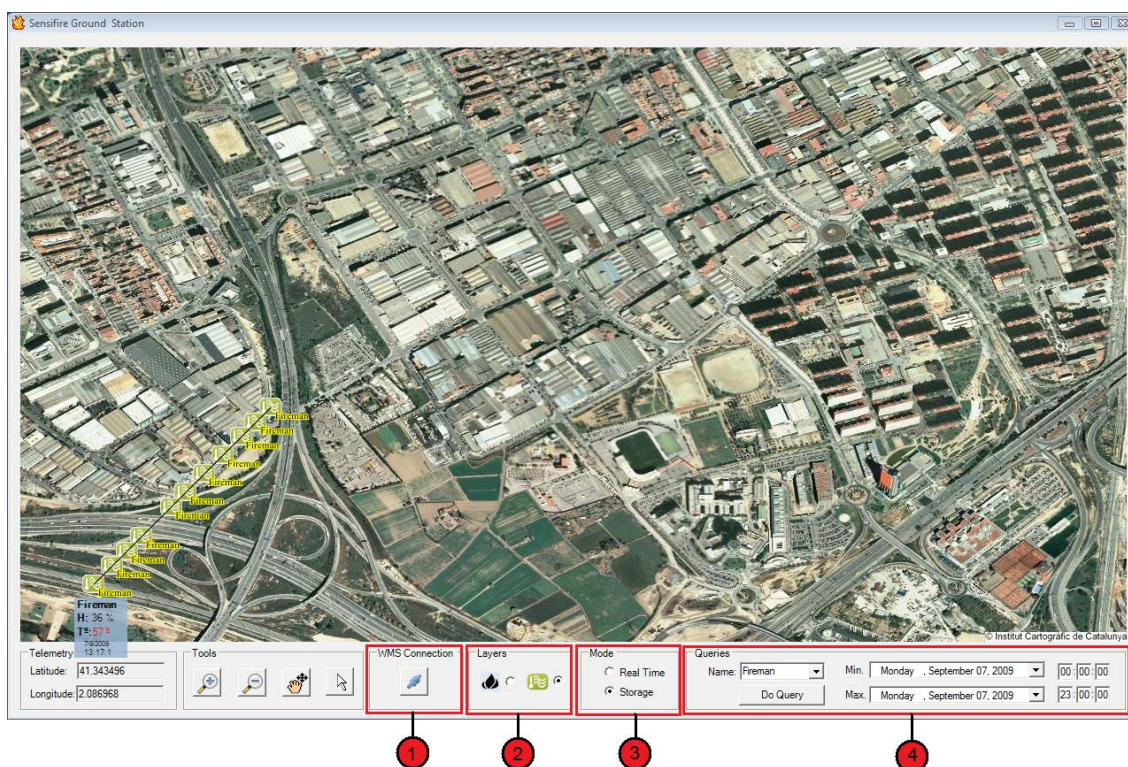


Fig. 6.8 Interfaz gráfica del servicio *GroundStation*

En el anexo 1 se presenta una guía con un ejemplo del funcionamiento de este servicio.

CAPÍTULO 7: CONCLUSIONES

7.1. Conclusiones tecnológicas

Con la elaboración de este proyecto se ha conseguido crear una versión inicial de un sistema que permita monitorizar a un escuadrón terrestre de bomberos. Una de las particularidades de este sistema es que utiliza *MAREA (Middleware Architecture for Remote Embedded Applications)*. Este middleware, nos ha permitido implementar el sistema de forma distribuida, basando su funcionamiento en servicios. Además se ha conseguido crear una versión compatible de este con plataforma *.NET Compact Framework*.

La implementación de software de este sistema se ha llevado a cabo gracias a la utilización de tecnologías como *SIG* o bases de datos orientadas a objetos. Se han utilizado librerías *SharpMap* y *Perst* ya que son de código abierto, con el fin de reducir al máximo los costes.

Por otra parte, para la implementación de hardware se ha utilizado la plataforma “open-hardware” *Arduino* que utiliza el lenguaje *Processing/Wiring*.

Recuperando los objetivos marcados para este proyecto en la introducción, y analizándolos nos encontramos que se han alcanzado gratamente:

- Se ha creado un sistema portátil de sensores que se puede incluir dentro del equipo de un bombero
- Se ha implementado un sistema distribuido con una arquitectura SOA (Arquitectura Orientada a Servicios) cuya comunicación se realiza a través del middleware MAREA.
- Se han utilizado tecnologías *SIG* (Sistemas de Información Geográfica) para controlar la cartografía y procesar los datos geográficos del sistema.
- Se ha creado un sistema sencillo que permite monitorizar la situación del bombero en primera persona y remotamente desde una estación central.
- Se ha implementado un sistema de almacenamiento de datos.

7.2. Impacto medioambiental

Uno de los objetivos de este proyecto es ofrecer una herramienta de apoyo para la extinción de incendios con el fin de reducir el impacto medioambiental de los mismos.

En la introducción de este documento se incluyen algunas de las consecuencias de los incendios en el ecosistema.

Por otro lado, todos los componentes electrónicos utilizados en la implementación de hardware de este proyecto cumplen con la normativa RoHS Compliant. Esta normativa restringe el uso de plomo, mercurio, cadmio, cromo hexavalente, así como los piorretardantes PBB (bifenilos polibromados) y PBDE (éter difenílico polibromado).

7.3. Conclusiones personales

La elaboración de este proyecto nos ha proporcionado la posibilidad real de desarrollar una aplicación desde el inicio hasta el fin. En este sentido, nos ha supuesto un reto personal muy enriquecedor.

Por otro lado, este proyecto nos ha ayudado a adquirir conocimientos en tecnologías que antes prácticamente desconocíamos como bases de datos orientadas a objetos y sistemas SIG.

Con la elaboración del mismo nos hemos dado cuenta que el desarrollo de un proyecto es algo bastante complejo que requiere un gran esfuerzo y dedicación pero sobretodo un estricto control de “*timings*” para poder cumplir los objetivos.

Además, la experiencia de trabajo con nuestro tutor ha sido positiva también, en el sentido en que teníamos libertad de acción. Hemos tenido una comunicación directa y personal que nos ha permitido cumplir con los objetivos y plazos.

7.4. Futuras líneas de trabajo

Al tratarse de una versión inicial, el sistema plantea varias futuras líneas de trabajo y mejoras.

A corto plazo, se debería plantear la representación de los incendios en el mapa. También se debería implementar la parte de hardware de la estación de central (anemómetro, veleta y micro controlador).

En la PDA del bombero se podría incorporar una interfaz SIG, para que este pudiera saber la posición y estado de sus compañeros, además de la suya. Esta funcionalidad se podría implementar gracias a *SharpMap* ya que ofrece soporte para el entorno *Compact Framework*.

También se podría incluir un servicio de comunicación de voz entre el bombero y la estación central, aprovechando el micrófono del que disponen las PDAs para grabar audio. Con esto se podría llegar a tener conversación similar a la de un walkie-talkie. Otra opción a contemplar sería la utilización de smartphones que permitieran la comunicación por la red telefónica móvil.

Una de las funcionalidades que también se contempló e incluso se empezó a implementar fue el envío de fotografías y video de la PDA a la estación central. Otro punto a estudiar sería la incorporación de un sistema que permitiera la monitorización de algunas de las constantes vitales del bombero (temperatura corporal, pulso, frecuencia respiratoria, presión arterial...)

Uno de los objetivos finales es llegar a unir este sistema con otro que contempla segmento aéreo. De este modo el segmento aéreo enviaría la información al equipo de tierra. Posteriormente este gestionaría los recursos terrestres para optimizar el control del incendio.

BIBLIOGRAFÍA

- [1] Ceballos Sierra, F. J., *Microsoft C#: Lenguaje y aplicaciones*, 2ª ED., Editorial RA-MA, 2007.
- [2] Rubio J.; Royo P.; Pastor E.; Barrado C.; Santamaria E. "A Middleware Architecture for Unmanned Aircraft Avionics.". Technical University of Catalonia - Computer Architecture Department, Spain.
- [3] Portal de referencia sobre Sistemas de Información Geográfica. Última consulta: 2 de Octubre 2009. Disponible en: <http://www.gabrielortiz.com/>
- [4] Open Source Geospatial Foundation. Última consulta: 4 de Septiembre 2009 Disponible en: <http://www.osgeo.org/>
- [5] SharpMap - Geospatial Application Framework for the CLR. Última consulta: 20 de Noviembre 2009 Disponible en: <http://www.codeplex.com/SharpMap>
- [6] Proj.NET. Última consulta: 2 de diciembre 2009. Disponible en: <http://www.codeplex.com/projnet>
- [7] Windows Mobile Native Thread Synchronization. Última consulta: 24 de Julio 2009. Disponible en: <http://www.codeproject.com/KB/mobile/WiMoNativeSync.aspx>
- [8] Using the GPS Intermediate Driver from Managed Code. Última consulta: 5 de Julio 2009. Disponible en: <http://msdn.microsoft.com/en-us/library/bb158708.aspx>
- [9] Harrington, Jan. L., *Object-Oriented Database Design Clearly Explained*, 1ª ED., Morgan Kaufman, 1999.
- [10] Perst Embedded Database. Última consulta: 22 de Diciembre 2009. Disponible en: <http://www.mcobject.com/perst>
- [11] McObject LLC. "Real-time Databases for Embedded Systems".[En línea]. Disponible en: <http://www.mcobject.com/index.cfm?fuseaction=download&pageid=469§ionid=130>.
- [12] OpenNETCF. Última consulta: 28 de Agosto 2009. Disponible en: <http://www.opennetcf.com/>
- [13] Arduino. Última consulta: 12 de diciembre 2009. Disponible en: <http://www.arduino.cc>

[14] Tanja, Radu, Cormac, Fay. "Wearable sensing applications: Carbon dioxide monitoring for emergency personnel using wearable sensors". *[En línea]*. Última consulta: 15 de enero 2010. Disponible en: <http://www.zarlink.com/zarlink/wearable-sensing-applications-whitepaper.pdf>

ANEXO 1. FUNCIONAMIENTO DEL SERVICIO GROUND STATION

En el siguiente anexo se presenta una pequeña guía de funcionamiento del servicio Ground Station. Para ver un ejemplo consideramos un incendio sobre un terreno cualquiera en el que se despliega un escuadrón terrestre del cuerpo de bomberos.



Fig. A1.1 Escenario general del incendio

Antes de distribuir a los miembros del escuadrón o escuadrones en la zona del incendio, cada uno de ellos deberá activar su módulo de bombero. Posteriormente deberán hacer las siguientes comprobaciones:

- Conectividad Bluetooth: El bombero deberá comprobar si la PDA de su módulo está correctamente asociada y conectada por Bluetooth al prototipo de hardware. La caja en la que está montado dispone de un LED azul que indica el estado de la conexión (Fig. A1.2). Además el bombero también podrá consultar el estado de esta en la interfaz gráfica del servicio Sensifire (Fig. A1.3).



Fig. A1.2 Prototipo de hardware del módulo de bombero

- Señal GPS: El bombero deberá comprobar la señal GPS que recibe es válida. Para ello deberá consultar, en el servicio Sensifire, la barra de estado de conexión con el dispositivo GPS de la PDA (Fig. A1.3).

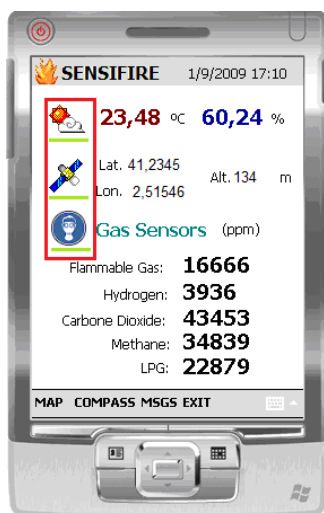


Fig. A1.3 Prototipo de hardware del módulo de bombero

Posteriormente un operador ejecutara el servicio Ground Station en la estación central. La estación central estará integrada en el interior de un camión del cuerpo de bomberos.

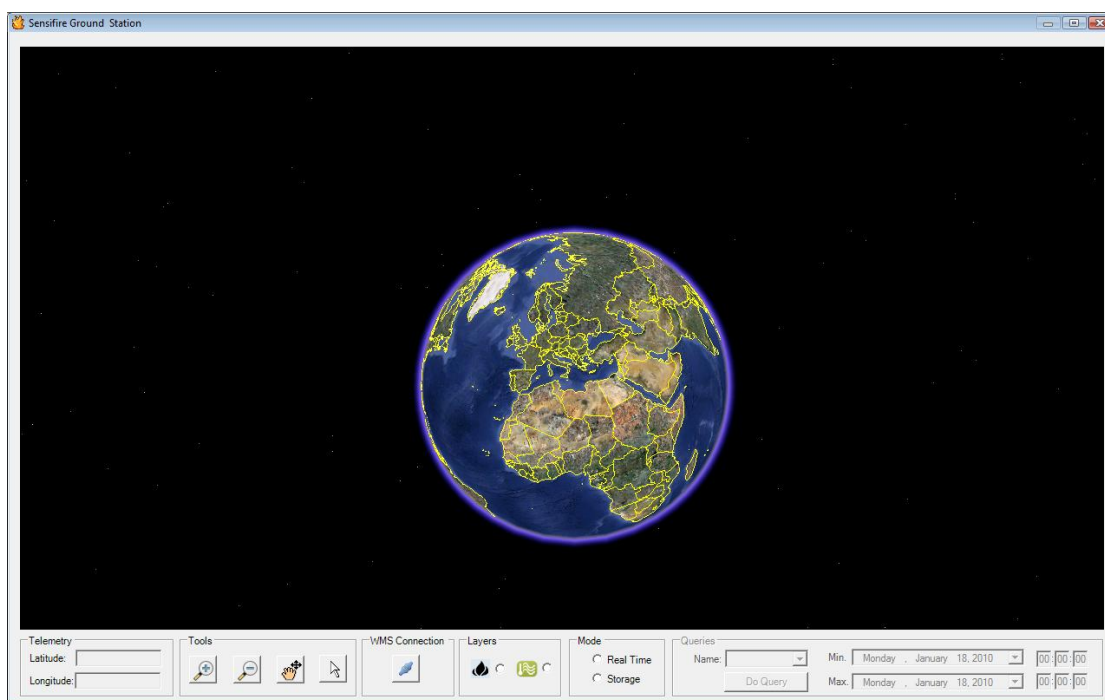


Fig. A1.4 Interfaz gráfica del servicio Ground Station

Lo primero que deberá hacer el operador después de iniciar el servicio es escoger el servidor WMS del que desea descargarse los mapas. Para hacerlo el programa dispone del botón “WMS Connection”. Al clicar el botón se abre un formulario que permite seleccionar el servidor, capas y posición por defecto en la que se carga el mapa (Fig. A1.5).

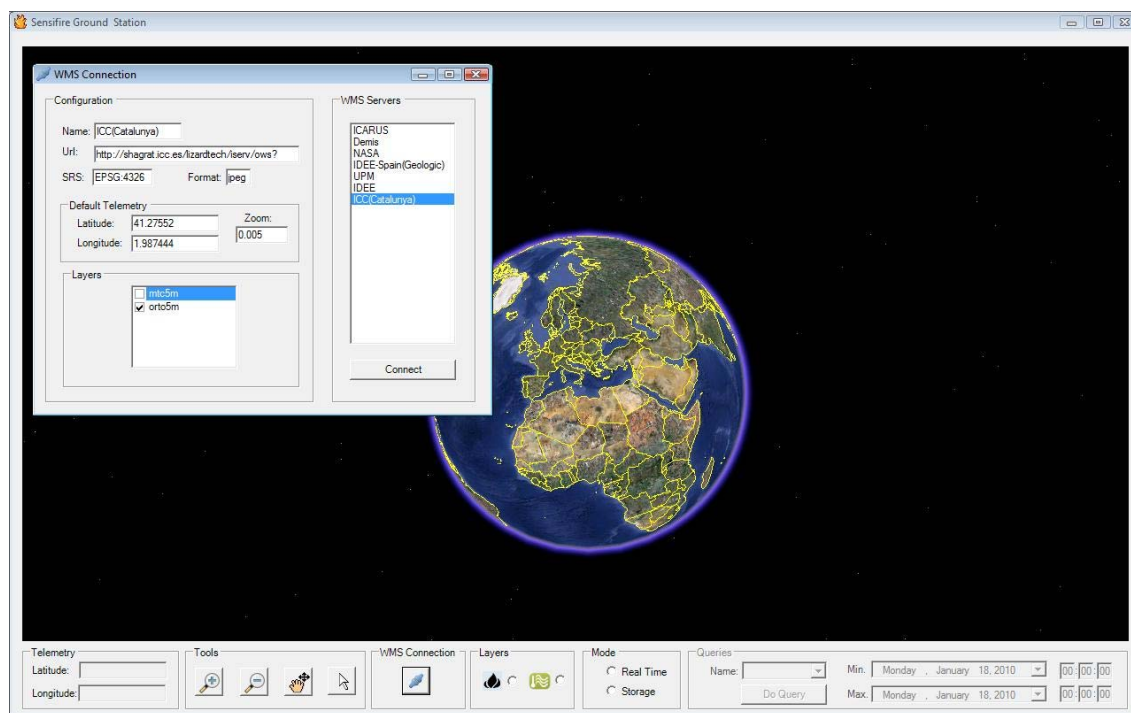


Fig. A1.5 Selección de servidor y capas WMS

A partir de ese momento el operador podrá visualizar en tiempo real el escuadrón de bomberos. Si desea consultar las medidas de los diferentes sensores de un bombero deberá situar el mouse encima del icono del bombero (Fig. A1.6).

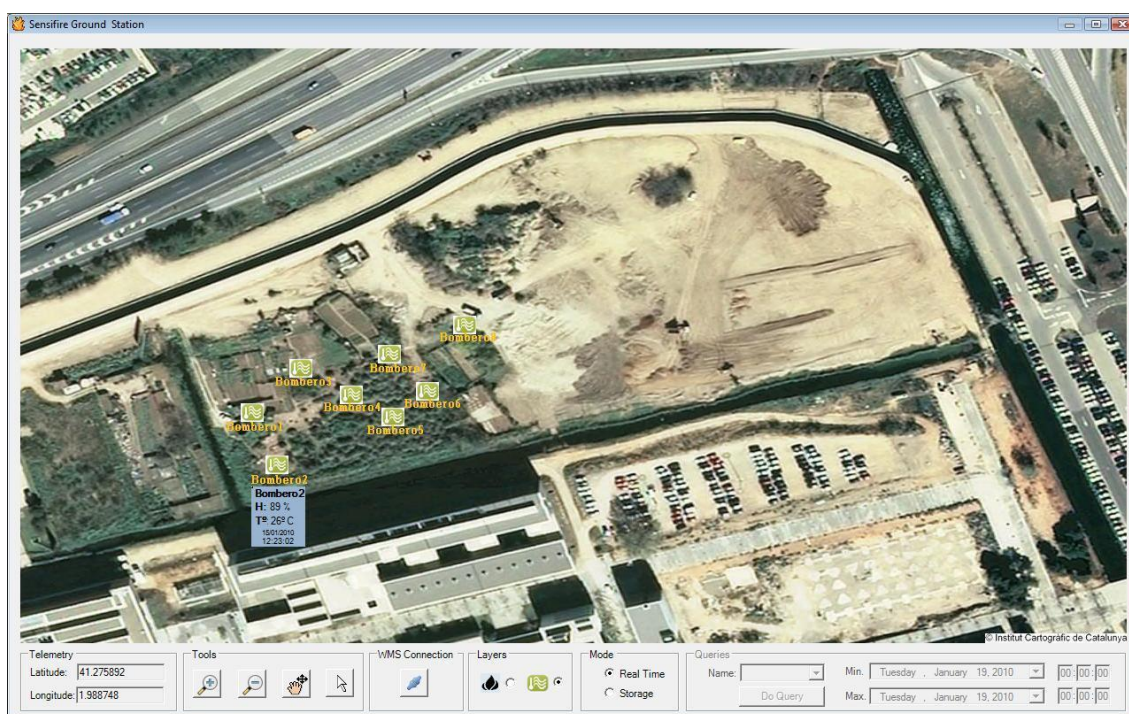


Fig. A1.6 Medidas medioambientales de los diferentes bomberos

El operador podrá consultar las medidas ambientales y de gas por separado. Este deberá seleccionar el tipo de medida que desea visualizar seleccionando la capa correspondiente en la zona "Layers".

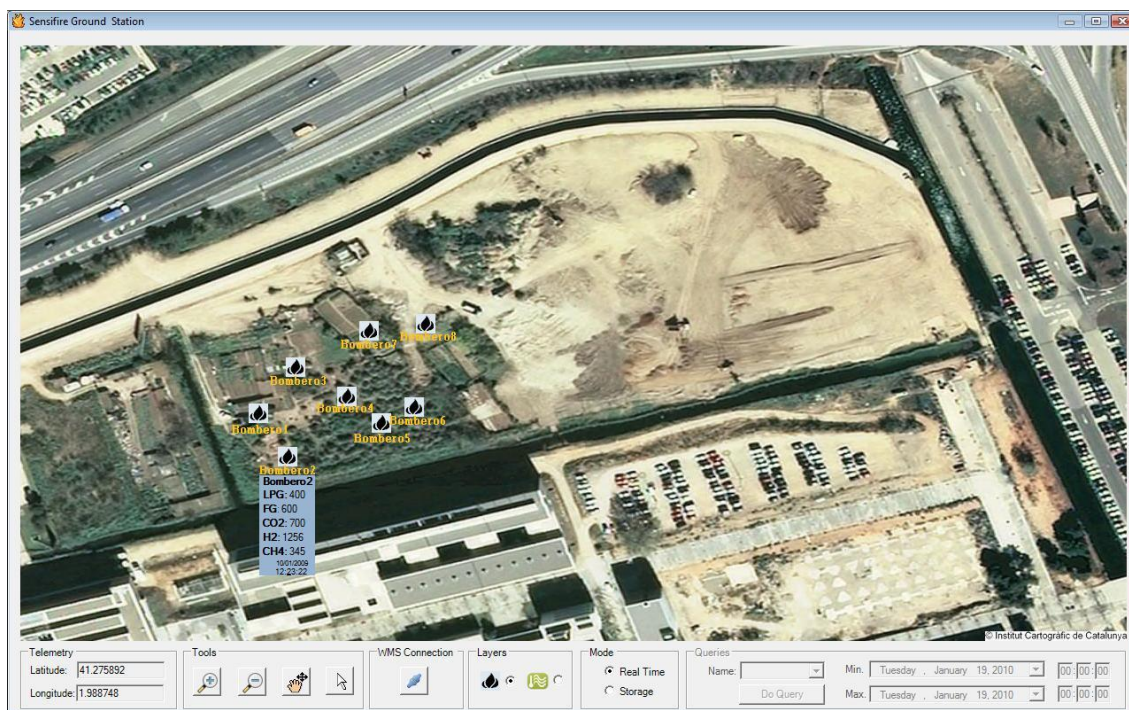


Fig. A1.7 Medidas gases de los diferentes bomberos

En caso de que algunas de las muestras obtenidas por los sensores indiquen que el bombero puede encontrarse ante una situación de peligro, el programa mostrará las alarmas correspondientes en la parte inferior derecha del mapa. En ellas se mostrará la hora desde que lleva activa la alarma, el identificador de bombero, el tipo de alarma (ambiental o gas) y el valor obtenido por el sensor. Además el valor de la medida del sensor, que se muestra al situar el mouse encima del icono del bombero, se mostrará en rojo.

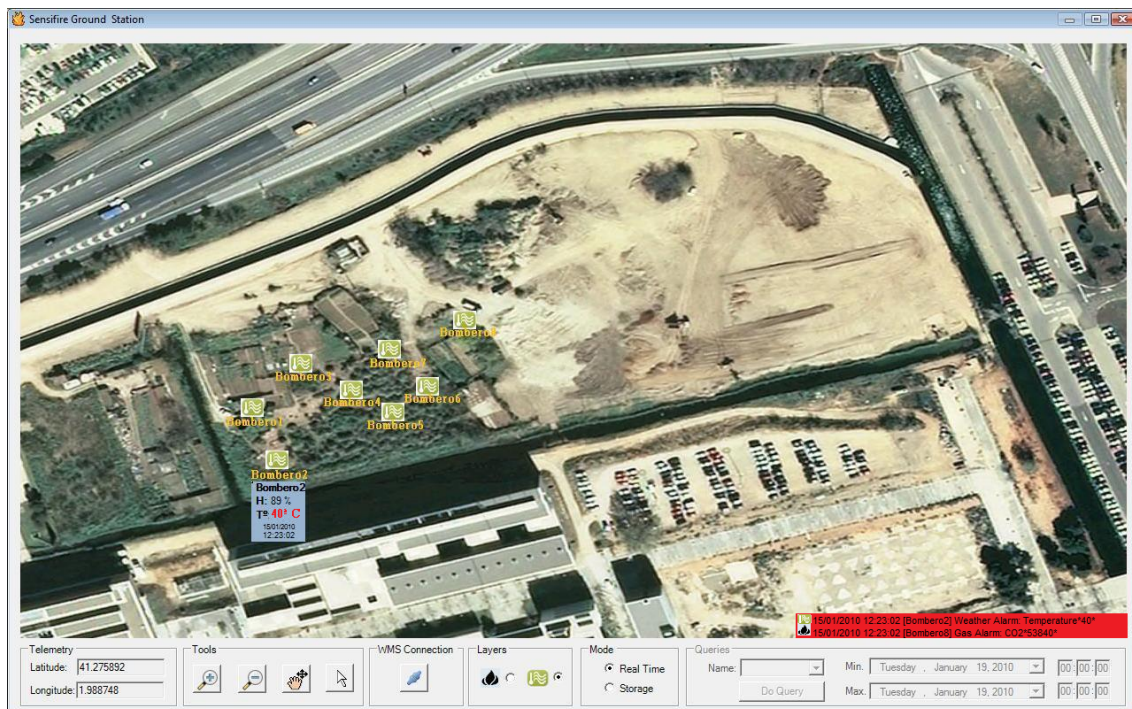


Fig. A1.8 Sistema de alarmas

El servicio Ground Station permite hacer un análisis a posteriori del histórico de medidas y recorrido del bombero. Para que funcione correctamente es necesario que el servicio Storage, que maneja la base de datos, este activado.

Para hacer una consulta se debe ejecutar el programa en modo almacenamiento ("Storage" de la zona "Mode"). El operador podrá hacer la consulta seleccionando el identificador del bombero en el desplegable "Name" de la zona "Queries". Posteriormente deberá introducir los intervalos de tiempo máximo y mínimo entre los que desea hacer la consulta.

Una vez realizada la consulta mediante el botón "Do query", el operador podrá visualizar el valor de las medidas tomadas por los diferentes sensores situando el mouse sobre las diferentes muestras del bombero (Fig. A1.9).

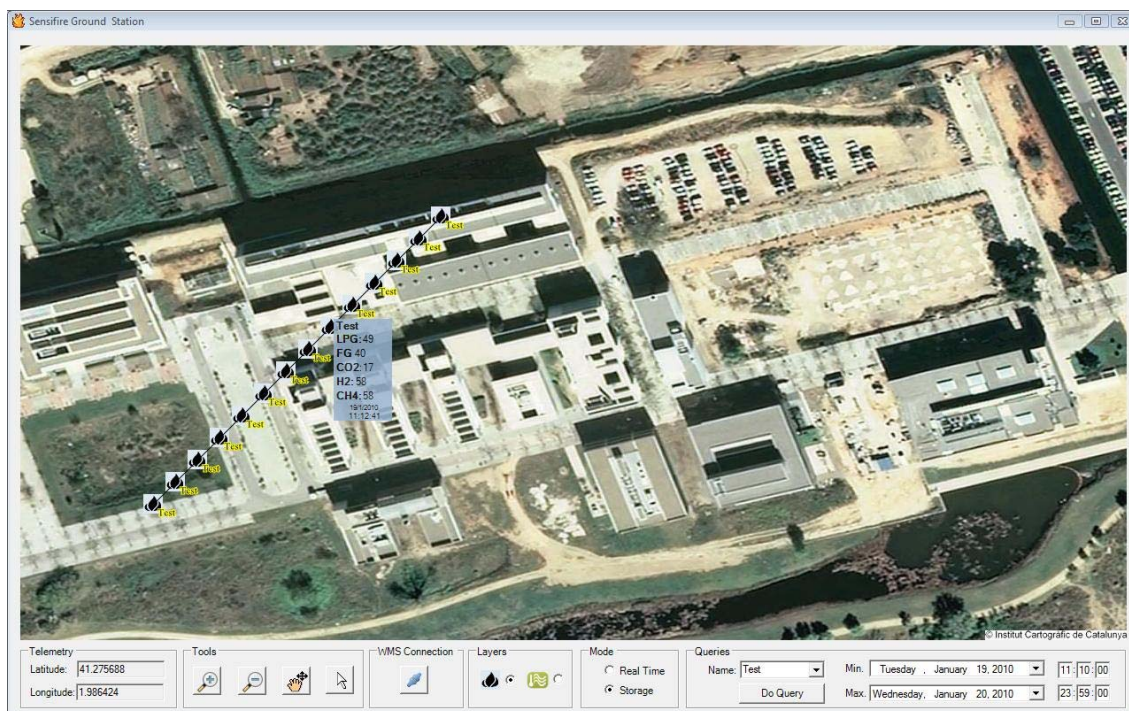


Fig. A1.9 Histórico del bombero (recorrido y medidas de gases)

De la misma manera que se hace en el modo en tiempo real, el operador podrá visualizar las medidas de gas o ambientales seleccionando el tipo de medida en la zona “Layers”.



Fig. A1.10 Histórico del bombero (recorrido y medidas ambientales)

ANEXO 2. MONTAJE DEL PROTOTIPO

En este anexo se muestran las fotos del prototipo de la caja que contiene los sensores y Arduino.

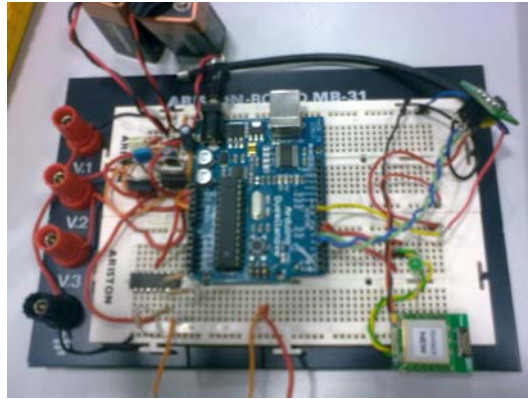


Fig. A2.1 Circuito montado en protoboard



Fig. A2.2 Caja de prototipado con sensor de SHT15

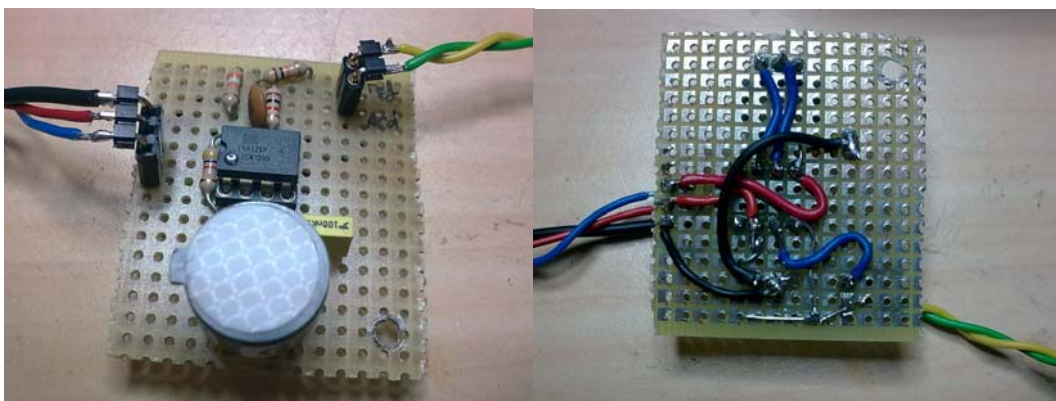


Fig. A2.3 Sensor de CO₂ con circuito de adaptación

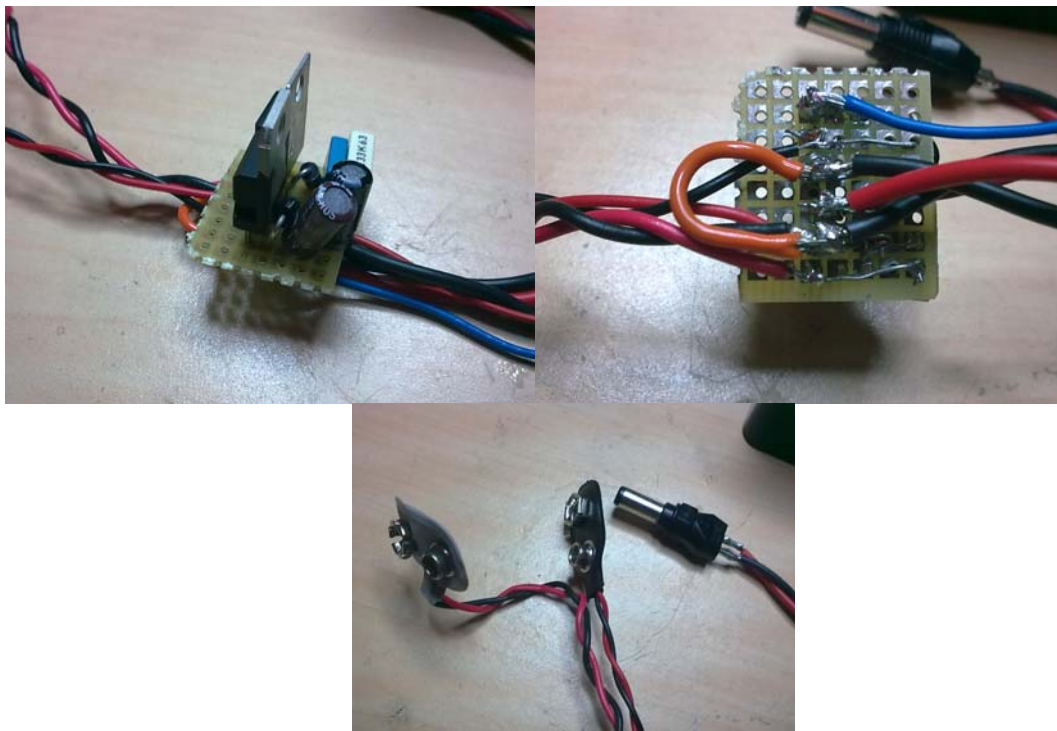


Fig. A2.4 Fuente de alimentación y conexiones

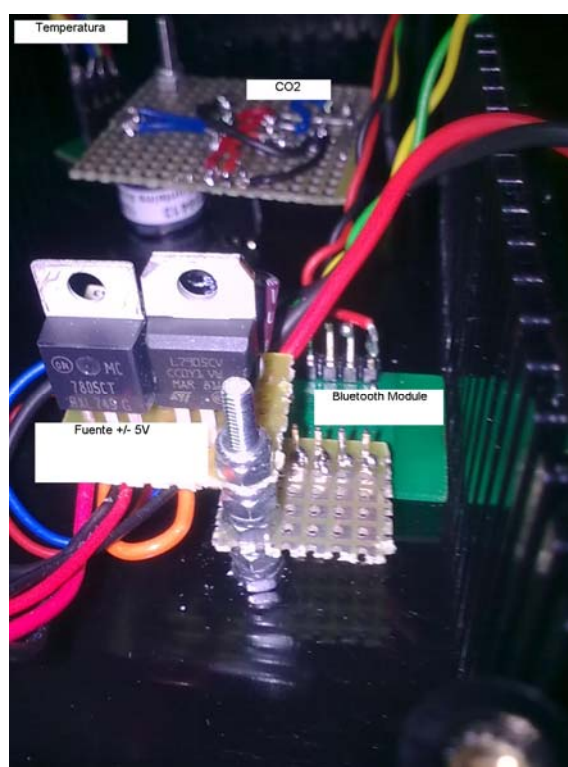


Fig. A2.5 Disposición de componentes dentro de la caja

ANEXO 3. INSTALACIÓN DRIVER MCL

Para instalar MCL en un PC se debe instalar primero el adaptador de red y seguidamente el protocolo. Los archivos se pueden descargar de la página de Microsoft Research. Posteriormente se debe instalar este mismo driver en una PDA. Concretamente se ha instalado una distribución para Windows Mobile 5.0. Una de las limitaciones de este driver es que su desarrollo se ha detenido, por lo que no existen versiones compatibles para Windows Mobile 6 o posteriores.

Para instalar el adaptador de red, debemos añadir el dispositivo manualmente (Fig. 3.1, Fig. 3.2), una vez añadido, comprobamos que este en “conexiones de red” (Fig. 3.3).



Fig. 3.1 “Añadir hardware” en “Panel de control”



Fig. 3.2 Capturas del asistente de instalación de hardware

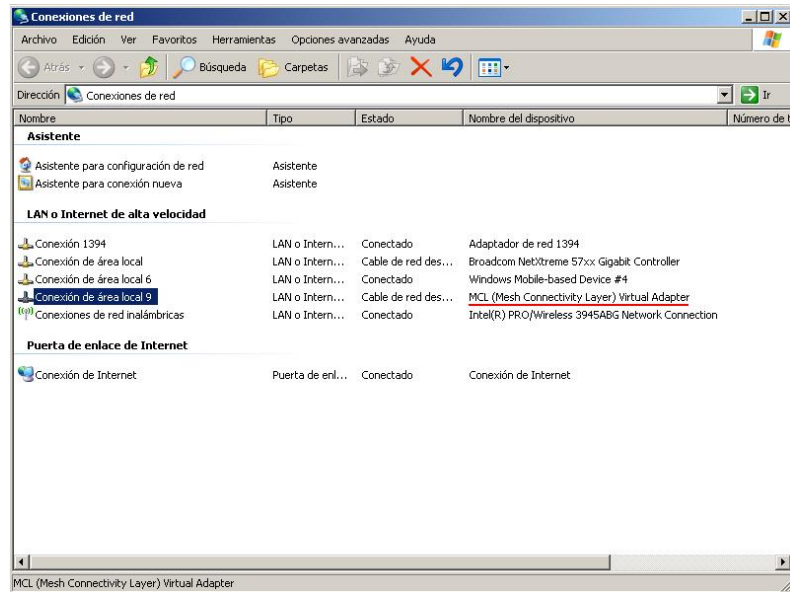


Figura 3.3. Comprobación que existe en conexiones de red

Instalamos el archivo que nos hemos descargado de la web para agregar el protocolo (Fig 3.4, 3.5, 3.6, 3.7).

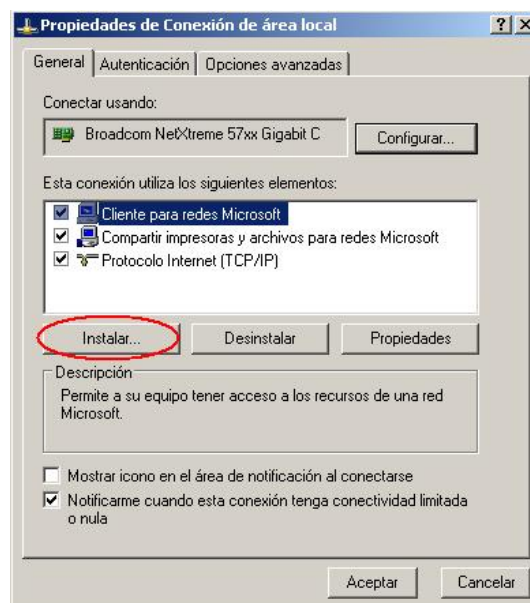


Fig. 3.4 Paso 1: Instalar...



Fig. 3.5 Paso 2: Agregar protocolo...



Fig. 3.6 Paso 3: Seleccionar MCL

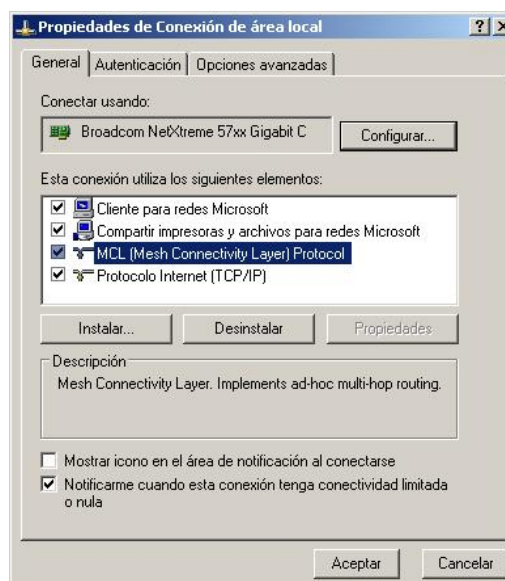


Fig. 3.7 Paso 4: Verificar la instalación correcta

Dependiendo de la versión del Windows Mobile 5.0 que tengamos (2003 o 2005) instalamos unos paquetes u otros (Fig 3.8, Fig. 3.9). Para que este se instale correctamente la conectividad WiFi debe estar apagada.

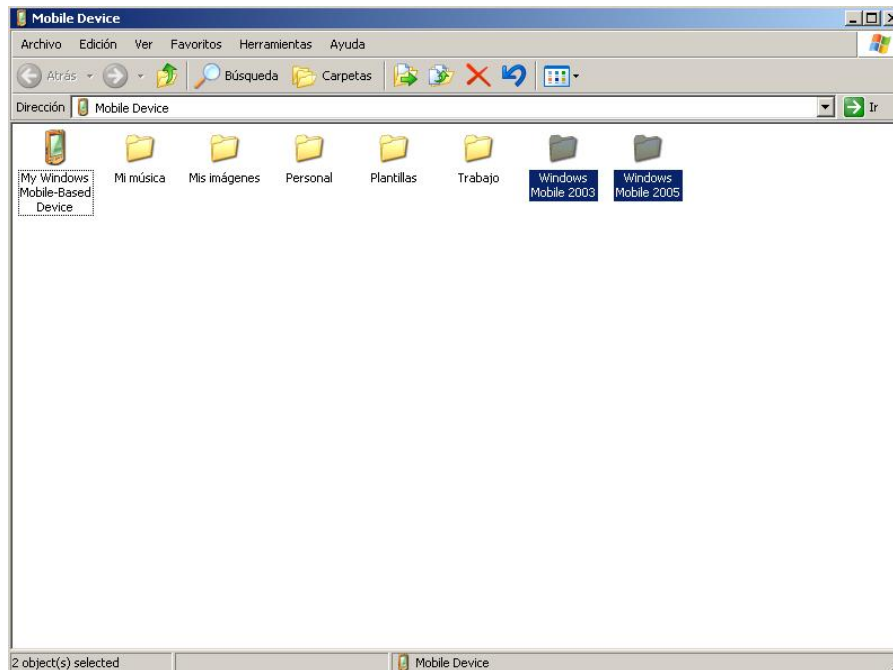


Fig. 3.8 Paquete de instalación para Mobile 2003 y 2005



Fig. 3.9 Archivos a instalar desde la PDA

ANEXO 4. CLASES

En el siguiente apartado se introducen las clases diseñadas específicamente para la implementación del sistema Sensifire.

Las clases Weather, Gas, Security y Position se utilizan para almacenar los datos obtenidos por los diferentes sensores y la posición del dispositivo GPS de la PDA.

```
public class Weather : Persistent
{
    public string Id;
    public double Temperature;
    public double Humidity;
    public ArduinoState State;
    public DateTime Time;
}

public class Gas : Persistent
{
    public string Id;
    public double Flammable_Gas;
    public double Hydrogen;
    public double Carbone_Dioxide;
    public double Methane;
    public double LPG;
    public DateTime Time;
}

public class Security : Persistent
{
    public string Id;
    public string Movement;
    public DateTime Time;
}

public class Position : Persistent
{
    public string Id;
    public double Latitude;
    public double Longitude;
    public double Altitude;
    public GPSState State;
    public DateTime Time;
}

public enum GPSState { Connected_without_GPS_signal,
NotConnected, Connected_with_GPS_signal };

public enum ArduinoState { Connected, NotConnected };
```

```
public class WeatherPosition
{
    public Weather weather;
    public Position position;
    public WeatherPosition(Weather wea, Position pos)
    {
        this.weather=wea;
        this.position=pos;
    }
}

public class GasPosition
{
    public Gas gas;
    public Position position;

    public GasPosition(Gas gas, Position pos)
    {
        this.gas=gas;
        this.position=pos;
    }
}
```

Los objetos de este tipo se utilizan en todo el sistema a excepción del modo de almacenamiento del servicio GroundStation. En este modo se hacen peticiones a una base de datos Perst que retorna las diferentes medidas del módulo del bombero referenciadas a la posición en que estas fueron tomadas (GasPosition, WeatherPosisiton).

ANEXO 5. LIBRERÍAS EXTERNAS

Como se ha comentado previamente durante la memoria se han utilizado únicamente librerías externas de código abierto.

Para la representación de los datos SIG se han utilizado librerías alojadas en “CodePlex”. Véase <http://codeplex.codeplex.com/>.

CodePlex (Microsoft), es un alojamiento de sitios web de proyectos de código abierto. Esta librería permite crear nuevos proyectos para compartir con el mundo, unirse a otros que ya han iniciado sus propios proyectos, o utilizar las aplicaciones en este sitio y proporcionar información. SharpMap, ProjNet son librerías alojadas en CodePlex.

Para el almacenamiento de datos se ha utilizado la librería Perst disponibles en <http://www.mcobject.com/perst>. Esta librería ha sido creada por McObject, empresa que proporciona tecnologías de bases de datos embebidas. McObject fue fundada por personas con una gran experiencia en los sistemas de bases de datos en tiempo real.

Para la obtención de coordenadas del dispositivo GPS de la PDA se ha utilizado la librería Microsoft.WindowsMobile.Samples.Location disponible en los ejemplos del SDK de Windows Mobile 6.

En los siguientes apartados se explican todas las librerías excepto Perst. El funcionamiento y descripción de esta librería se ha especificado en el capítulo 3 de la memoria.

A5.1. SharpMap

SharpMap es una librería para facilitar el trabajo con mapas para su uso en web y aplicaciones de escritorio. El motor está escrito en C # y basado en el .NET 2.0 Framework. SharpMap es liberado bajo la GNU Lesser General Public License.

Formatos de datos soportados:

Vector data:

- ESRI Shape files format
- PostgreSQL/PostGIS
- OLEDB (points only)
- Microsoft SQL Server
- Oracle
- GPX
- MapInfo File

- TIGER
- S57
- DGN
- CSV
- GML
- Interlis 1
- Interlis 2
- SQLite" and "ODBC"

Raster data :

- Arc/Info ASCII Grid
- Arc/Info Binary Grid (.adf)
- Microsoft Windows Device Independent Bitmap (.bmp)
- BSB Nautical Chart Format (.kap)
- VTP Binary Terrain Format (.bt)
- CEOS (Spot for instance)
- First Generation USGS DOQ (.doq)
- DODS / OPeNDAP
- New Labelled USGS DOQ (.doq)
- Military Elevation Data (.dt0, .dt1)
- ERMapper Compressed Wavelets (.ecw)
- ESRI .hdr Labelled
- ENVI .hdr Labelled Raster
- Envisat Image Product (.n1)
- EOSAT FAST Format
- FITS (.fits)
- Graphics Interchange Format (.gif)
- GMT Compatible netCDF
- GRASS Rasters
- TIFF / GeoTIFF (.tif)
- Hierarchical Data Format Release 5 (HDF5)
- Erdas Imagine (.img)
- Idrisi Raster
- Image Display and Analysis (WinDisp)
- ILWIS Raster Map (.mpr, .mpl)
- Japanese DEM (.mem)
- JPEG JFIF (.jpg)
- NOAA Polar Orbiter Level 1b Data Set (AVHRR)
- Erdas 7.x .LAN and .GIS
- Multi-resolution Seamless Image Database
- Meteosat Second Generation
- NDF
- NITF
- OGD Bridge
- PCI Geomatics Database File
- Portable Network Graphics (.png)
- PCRaster (.map)
- Netpbm (.ppm, .pgm)

- Swedish Grid RIK (.rik)
- RadarSat2 XML (product.xml)
- USGS SDTS DEM (CATD.DDF)
- Raster Matrix Format (.rsw, .mtw)
- SGI Image Format
- USGS ASCII DEM (.dem)
- X11 Pixmap (.xpm)

Modelo de los objetos GEOMETRIA que dispone:

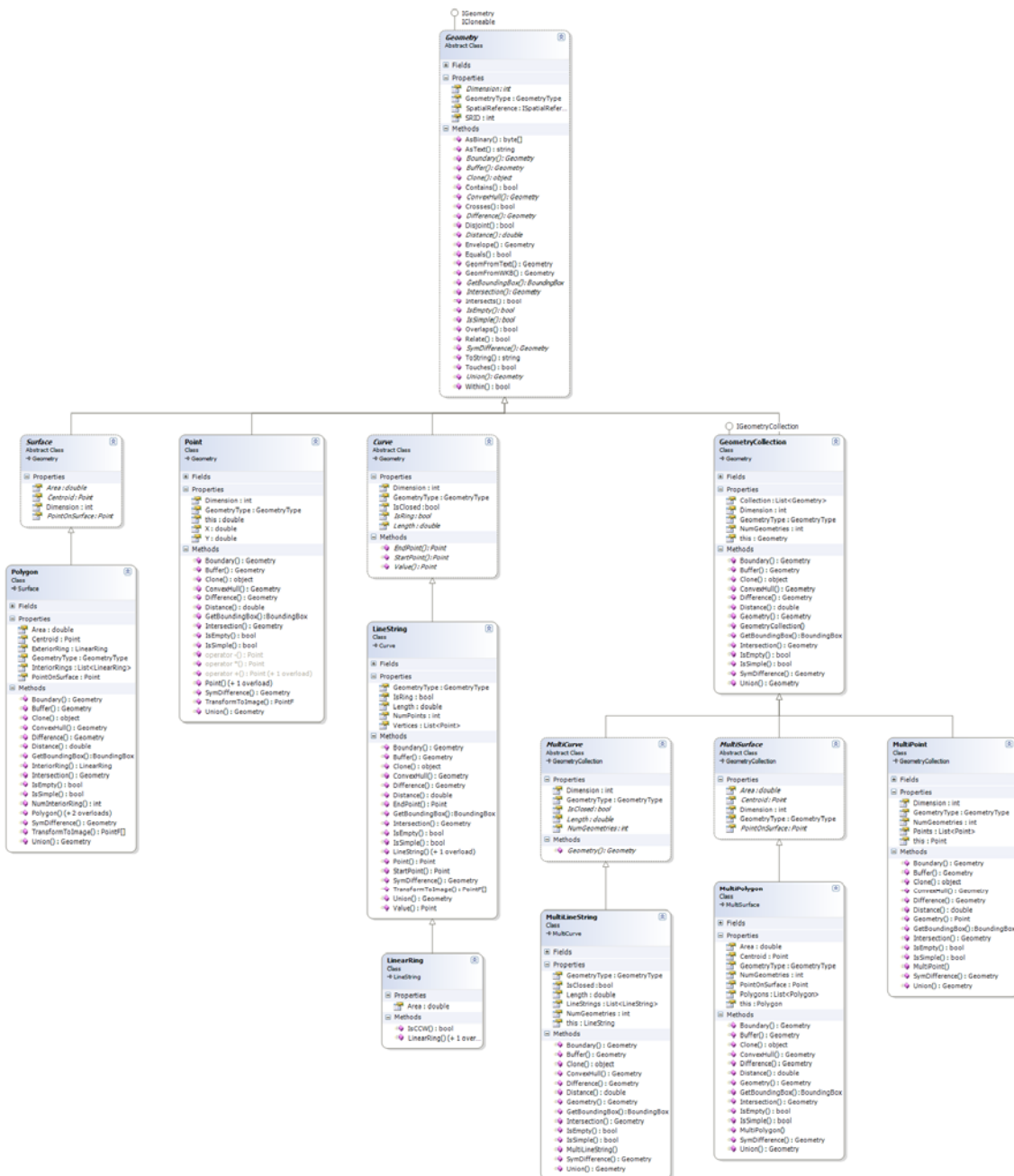


Fig. A5.1 Modelos de los objetos Geometry de SharpMap

Actualmente la versión de SharpMap estable es la 0.9, aunque actualmente están trabajando para acabar de formalizar la versión 2.0 y que ya se puede utilizar. Para darle más robustez a la aplicación que utiliza esta librería se optó por utilizar la versión estable de SharpMap.

Para más información consultar la página del proyecto:

<http://sharpmap.codeplex.com/>.

A5.2. SharpMap.UI

Esta librería forma parte de la misma librería y del mismo proyecto que SharpMap, está compuesta de los métodos y de los controles para poder visualizar el mapa. En el proyecto se usa un control “mapImage” para visualizar el mapa. Y gracias a esta librería obtenemos muchas variables extraídas directamente del mapa y los métodos para manejarlo.

Cabe puntualizar que también se utilizó otra versión del control “MapImage” llamado “MapBox” que fue desarrollado por las personas que dan soporte a SharpMap. Este control aseguraban que aumentada la efectividad del mapa, pero al probarlo no se percató ninguna diferencia por tanto se mantuvo la versión original de este control.

A5.3. Proj.NET

Proj.NET realiza conversiones geodésicas punto a punto geodésico entre sistemas de coordenadas para su uso en Sistemas de Información Geográfica (SIG) o aplicaciones GPS. Esta librería está totalmente integrada dentro de SharpMap como “SharpMap.Coordinates”. Pero esta implementada y tratada como un proyecto aparte.

Nos proporciona las siguientes características:

- Datum transformations
- Geographic, Geocentric, and Projected coordinate systems
- Compatible with both Microsoft .NET 2.0, Mono, .NET Compact Framework & Silverlight
- Converts coordinate systems to/from Well-Known Text (WKT) and to XML

Tipos de proyecciones que soporta:

- Mercator
- Transverse Mercator
- Albers
- Lambert Conformal
- Krovak

Para más información consultar la página del proyecto:

<http://projnet.codeplex.com/>.

A5.4. Microsoft.WindowsMobile.Samples.Location

Esta librería proporciona una capa que permite a las aplicaciones funcionar con el hardware GPS de un dispositivo móvil. El código fuente de esta librería se encuentra en los ejemplos del SDK de Windows Mobile 6 profesional.

Esta librería dispone del objeto Gps que nos permite acceder a dicho dispositivo. Además dispone de los objetos GpsState y GpsPosition que nos permiten obtener información del estado del GPS y su posición.

Observando los atributos de la clase Gps, vemos que dispone de algunos eventos como LocationChanged o DeviceStateChanged que sirven para notificar un cambio de ubicación o estado del dispositivo GPS.

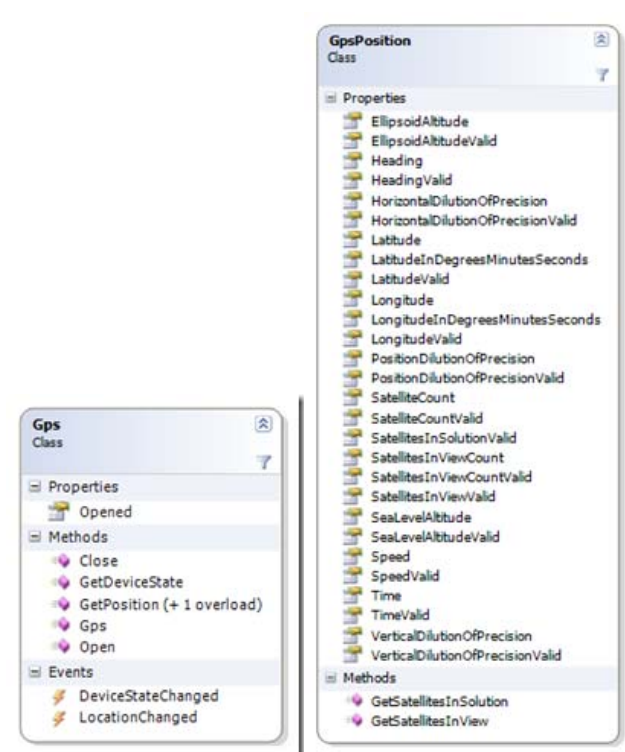


Fig. A5.2 Clase GPS Y GPS Position de la librería Microsoft.WindowsMobile.Samples.Location

ANEXO 6. DATASHEETS

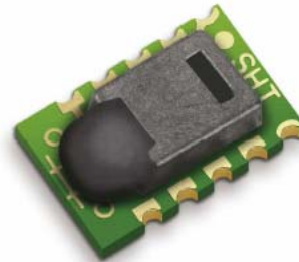
A6.1. Sensor de temperatura y humedad: SHT15

SENSIRION
THE SENSOR COMPANY

Datasheet SHT1x (SHT10, SHT11, SHT15)

Humidity and Temperature Sensor

- Fully calibrated
- Digital output
- Low power consumption
- Excellent long term stability
- SMD type package – reflow solderable



Product Summary

SHT1x (including SHT10, SHT11 and SHT15) is Sensirion's family of surface mountable relative humidity and temperature sensors. The sensors integrate sensor elements plus signal processing on a tiny foot print and provide a fully calibrated digital output. A unique capacitive sensor element is used for measuring relative humidity while temperature is measured by a band-gap sensor. The applied CMOSens® technology guarantees excellent reliability and long term stability. Both sensors are seamlessly coupled to a 14bit analog to digital converter and a serial interface circuit. This results in superior signal quality, a fast response time and insensitivity to external disturbances (EMC).

Each SHT1x is individually calibrated in a precision humidity chamber. The calibration coefficients are programmed into an OTP memory on the chip. These coefficients are used to internally calibrate the signals from the sensors. The 2-wire serial interface and internal voltage regulation allows for easy and fast system integration. The tiny size and low power consumption makes SHT1x the ultimate choice for even the most demanding applications.

SHT1x is supplied in a surface-mountable LCC (Leadless Chip Carrier) which is approved for standard reflow soldering processes. The same sensor is also available with pins (SHT7x) or on flex print (SHTA1).

Dimensions

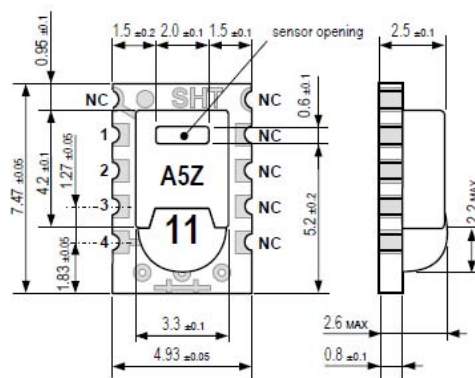


Figure 1: Drawing of SHT1x sensor packaging, dimensions in mm (1mm = 0.039inch). Sensor label gives "11" for SHT11 as an example. Contacts are assigned as follows: 1:GND, 2:DATA, 3:SCK, 4:VDD.

Sensor Chip

SHT1x V4 – for which this datasheet applies – features a version 4 Silicon sensor chip. Besides the humidity and temperature sensors the chip contains an amplifier, A/D converter, OTP memory and a digital interface. V4 sensors can be identified by the alpha-numeric traceability code on the sensor cap – see example "A5Z" code on Figure 1.

Material Contents

While the sensor is made of a CMOS chip the sensor housing consists of an LCP cap with epoxy glob top on an FR4 substrate. The device is fully RoHS and WEEE compliant, thus it is free of Pb, Cd, Hg, Cr(6+), PBB and PBDE.

Evaluation Kits

For sensor trial measurements, for qualification of the sensor or even experimental application of the sensor there is an evaluation kit *EK-H2* available including sensor, hard and software to interface with a computer.

For more sophisticated and demanding measurements a multi port evaluation kit *EK-H3* is available which allows for parallel application of up to 20 sensors.

Sensor Performance

Relative Humidity

Parameter	Condition	min	typ	max	Units
Resolution ¹		0.4	0.05	0.05	%RH
		8	12	12	bit
Accuracy ² SHT10	typical		±4.5		%RH
	maximal	see Figure 2			
Accuracy ² SHT11	typical		±3.0		%RH
	maximal	see Figure 2			
Accuracy ² SHT15	typical		±2.0		%RH
	maximal	see Figure 2			
Repeatability			±0.1		%RH
Replacement		fully interchangeable			
Hysteresis			±1		%RH
Nonlinearity	raw data		±3		%RH
	linearized		<<1		%RH
Response time ³	τ (63%)		8		s
Operating Range		0		100	%RH
Long term drift ⁴	normal		< 0.5		%RH/yr

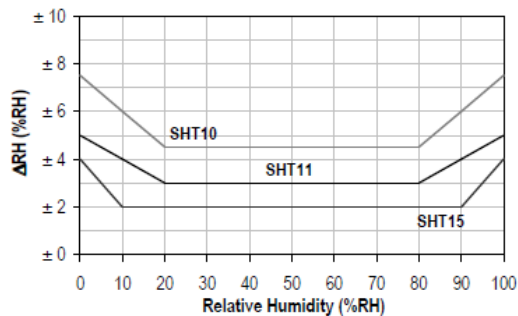


Figure 2: Maximal RH-accuracy at 25°C per sensor type.

Temperature

Parameter	Condition	min	typ	max	Units
Resolution ¹		0.04	0.01	0.01	°C
		12	14	14	bit
Accuracy ² SHT10	typical		±0.5		°C
	maximal	see Figure 3			
Accuracy ² SHT11	typical		±0.4		°C
	maximal	see Figure 3			
Accuracy ² SHT15	typical		±0.3		°C
	maximal	see Figure 3			
Repeatability			±0.1		°C
Replacement		fully interchangeable			
Operating Range		-40		123.8	°C
		-40		254.9	°F
Response Time ⁶	τ (63%)	5		30	s
Long term drift			< 0.04		°C/yr

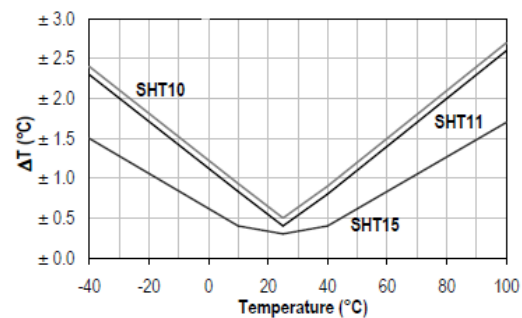


Figure 3: Maximal T-accuracy per sensor type.

Electrical and General Items

Parameter	Condition	min	typ	max	Units
Source Voltage		2.4	3.3	5.5	V
Power Consumption ⁵	sleep		2	5	μW
	measuring		3		mW
	average		150		μW
Communication	digital 2-wire interface, see Communication				
Storage	10 – 50°C (0 – 125°C peak), 20 – 60%RH				

Packaging Information

Sensor Type	Packaging	Quantity	Order Number
SHT10	Tape & Reel	2000	1-100218-04
SHT11	Tape & Reel	100	1-100051-04
	Tape & Reel	400	1-100098-04
	Tape & Reel	2000	1-100524-04
SHT15	Tape & Reel	100	1-100085-04
	Tape & Reel	400	1-100093-04

¹ The default measurement resolution of is 14bit for temperature and 12bit for humidity. It can be reduced to 12/8bit by command to status register.

² Accuracies are tested at Outgoing Quality Control at 25°C (77°F) and 3.3V. Values exclude hysteresis and non-linearity.

³ Time for reaching 63% of a step function, valid at 25°C and 1 m/s airflow.

⁴ Value may be higher in environments with high contents of volatile organic compounds. See Section 1.3 of Users Guide.

⁵ Values for VDD=5.5V at 25°C, average value at one 12bit measurement per second.

⁶ Response time depends on heat capacity of and thermal resistance to sensor substrate.

Users Guide SHT1x

1 Application Information

1.1 Operating Conditions

Sensor works stable within recommended normal range – see Figure 4. Long term exposures to conditions outside normal range, especially at humidity >80%RH, may temporarily offset the RH signal (+3 %RH after 60h). After return to normal range it will slowly return towards calibration state by itself. See Section 1.4 “Reconditioning Procedure” to accelerate eliminating the offset. Prolonged exposure to extreme conditions may accelerate ageing.

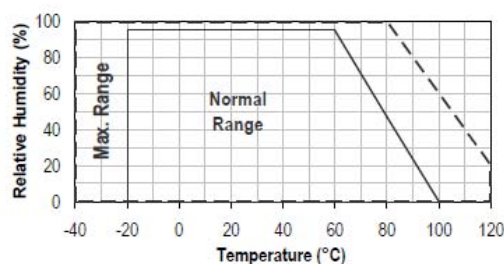


Figure 4: Operating Conditions

1.2 Soldering instructions

For soldering SHT1x standard reflow soldering ovens may be used. The sensor is qualified to withstand soldering profile according to IPC/JEDEC J-STD-020C with peak temperatures at 260°C during up to 40sec including Pb-free assembly in IR/Convection reflow ovens.

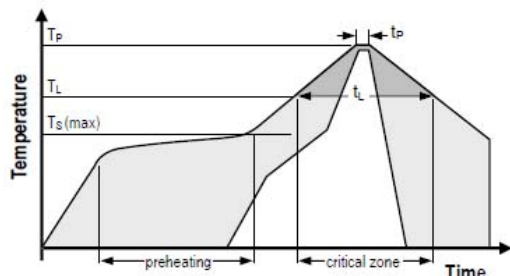


Figure 5: Soldering profile according to JEDEC standard. $T_P \leq 260^\circ\text{C}$ and $t_P < 40\text{sec}$ for Pb-free assembly. $T_L < 220^\circ\text{C}$ and $t_L < 150\text{sec}$. Ramp-up/down speeds shall be $< 5^\circ\text{C/sec}$.

For soldering in Vapor Phase Reflow (VPR) ovens the peak conditions are limited to $T_P < 233^{\circ}\text{C}$ during $t_P < 60\text{sec}$ and ramp-up/down speeds shall be limited to $10^{\circ}\text{C}/\text{sec}$. For manual soldering contact time must be limited to 5 seconds at up to 350°C .

7 $233^{\circ}\text{C} = 451^{\circ}\text{F}$, $260^{\circ}\text{C} = 500^{\circ}\text{F}$, $350^{\circ}\text{C} = 662^{\circ}\text{F}$

IMPORTANT: After soldering the devices should be stored at >75%RH for at least 12h to allow the polymer to re-hydrate. Otherwise the sensor may read an offset that slowly disappears if exposed to ambient conditions.

In no case, neither after manual nor reflow soldering, a board wash shall be applied. Therefore it is strongly recommended to use "no-clean" solder paste. In case of application with exposure of the sensor to corrosive gases the soldering pads shall be sealed to prevent loose contacts or short cuts.

For the design of the SHT1x footprint it is recommended to use dimensions according to Figure 7. Sensor pads are coated with 35µm Cu, 5µm Ni and 0.1µm Au.

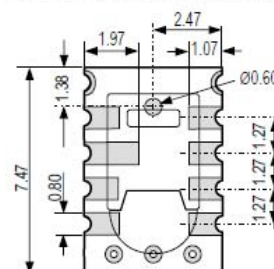


Figure 6: Rear side electrodes of sensor, view from top side.

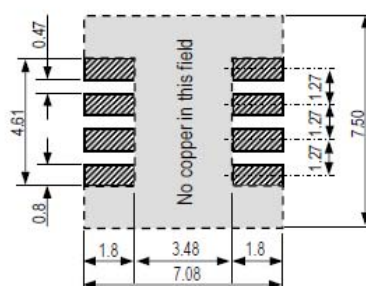


Figure 7: Recommended footprint for SHT1x. Values in mm.

1.3 Storage Conditions and Handling Instructions

It is of great importance to understand that a humidity sensor is not a normal electronic component and needs to be handled with care. Chemical vapors at high concentration in combination with long exposure times may offset the sensor reading.

For these reasons it is recommended to store the sensors in original packaging including the sealed ESD bag at following conditions: Temperature shall be in the range of 10°C – 50°C (0 – 125°C for limited time) and humidity at 20 – 60%RH (sensors that are not stored in ESD bags).

For sensors that have been removed from the original packaging we recommend to store them in ESD bags made of PE-HD⁸.

In manufacturing and transport the sensors shall be prevented of high concentration of chemical solvents and long exposure times. Out-gassing of glues, adhesive tapes and stickers or out-gassing packaging material such as bubble foils, foams, etc. shall be avoided. Manufacturing area shall be well ventilated.

For more detailed information please consult the document "Handling Instructions" or contact Sensirion.

1.4 Reconditioning Procedure

As stated above extreme conditions or exposure to solvent vapors may offset the sensor. The following reconditioning procedure may bring the sensor back to calibration state:

Baking: 100 – 105°C at < 5%RH for 10h
 Re-Hydration: 20 – 30°C at ~ 75%RH for 12h⁹.

1.5 Temperature Effects

Relative humidity reading strongly depends on temperature. Therefore, it is essential to keep humidity sensors at the same temperature as the air of which the relative humidity is to be measured. In case of testing or qualification the reference sensor and test sensor must show equal temperature to allow for comparing humidity readings.

If the SHT1x shares a PCB with electronic components that produce heat it should be mounted in a way that prevents heat transfer or keeps it as low as possible. Measures to reduce heat transfer can be ventilation, reduction of copper layers between the SHT1x and the rest of the PCB or milling a slit into the PCB around the sensor (see Figure 8).

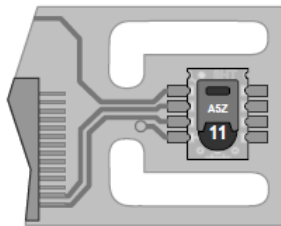


Figure 8: Top view of example of mounted SHT1x with slits milled into PCB to minimize heat transfer.

Furthermore, there are self-heating effects in case the measurement frequency is too high. Please refer to Section 3.3 for detailed information.

1.6 Light

The SHT1x is not light sensitive. Prolonged direct exposure to sunshine or strong UV radiation may age the housing.

1.7 Membranes

SHT1x does not contain a membrane at the sensor opening. However, a membrane may be added to prevent dirt and droplets from entering the housing and to protect the sensor. It will also reduce peak concentrations of chemical vapors. For optimal response times the air volume behind the membrane must be kept minimal. Sensirion recommends and supplies the SF1 filter cap for optimal IP54 protection (for higher protection – i.e. IP67 - SF1 must be sealed to the PCB with epoxy). Please compare Figure 9.

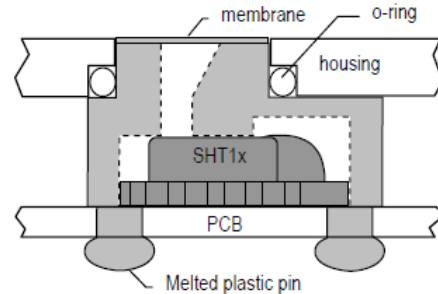


Figure 9: Side view of SF1 filter cap mounted between PCB and housing wall. Volume below membrane is kept minimal.

1.8 Materials Used for Sealing / Mounting

Many materials absorb humidity and will act as a buffer increasing response times and hysteresis. Materials in the vicinity of the sensor must therefore be carefully chosen. Recommended materials are: Any metals, LCP, POM (Delrin), PTFE (Teflon), PE, PEEK, PP, PB, PPS, PSU, PVDF, PVF.

For sealing and gluing (use sparingly): Use high filled epoxy for electronic packaging (e.g. glob top, underfill), and Silicone. Out-gassing of these materials may also contaminate the SHT1x (see Section 1.3). Therefore try to add the sensor as a last manufacturing step to the assembly, store the assembly well ventilated after manufacturing or bake at >50°C for 24h to outgas contaminants before packing.

1.9 Wiring Considerations and Signal Integrity

Carrying the SCK and DATA signal parallel and in close proximity (e.g. in wires) for more than 10cm may result in cross talk and loss of communication. This may be resolved by routing VDD and/or GND between the two data signals and/or using shielded cables. Furthermore, slowing down SCK frequency will possibly improve signal integrity. Power supply pins (VDD, GND) must be

⁸ For example, 3M antistatic bag, product "1910" with zipper.

⁹ 75%RH can conveniently be generated with saturated NaCl solution. 100 – 105°C correspond to 212 – 221°F, 20 – 30°C correspond to 68 – 86°F

decoupled with a 100nF capacitor if wires are used. Capacitor should be placed as close to the sensor as possible. Please see the Application Note "ESD, Latch-up and EMC" for more information.

1.10 ESD (Electrostatic Discharge)

ESD immunity is qualified according to MIL STD 883E, method 3015 (Human Body Model at ± 2 kV).

Latch-up immunity is provided at a force current of ± 100 mA with $T_{amb} = 80^{\circ}\text{C}$ according to JEDEC78A. See Application Note "ESD, Latch-up and EMC" for more information.

2 Interface Specifications

Pin	Name	Comment
1	GND	Ground
2	DATA	Serial Data, bidirectional
3	SCK	Serial Clock, input only
4	VDD	Source Voltage
NC	NC	Must be left unconnected

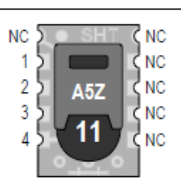


Table 1: SHT1x pin assignment, NC remain floating.

2.1 Power Pins (VDD, GND)

The supply voltage of SHT1x must be in the range of 2.4 – 5.5V, recommended supply voltage is 3.3V. Power supply pins Supply Voltage (VDD) and Ground (GND) must be decoupled with a 100 nF capacitor – see Figure 10.

The serial interface of the SHT1x is optimized for sensor readout and effective power consumption. The sensor cannot be addressed by I²C protocol; however, the sensor can be connected to an I²C bus without interference with other devices connected to the bus. The controller must switch between the protocols.

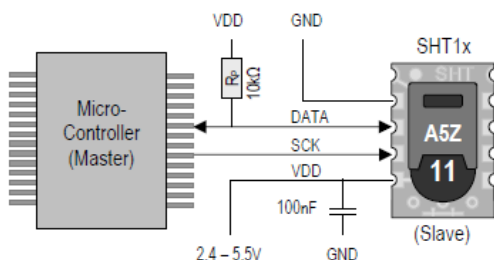


Figure 10: Typical application circuit, including pull up resistor R_P and decoupling of VDD and GND by a capacitor.

2.2 Serial clock input (SCK)

SCK is used to synchronize the communication between microcontroller and SHT1x. Since the interface consists of fully static logic there is no minimum SCK frequency.

2.3 Serial data (DATA)

The DATA tri-state pin is used to transfer data in and out of the sensor. For sending a command to the sensor, DATA is valid on the rising edge of the serial clock (SCK) and must remain stable while SCK is high. After the falling edge of SCK the DATA value may be changed. For safe communication DATA valid shall be extended T_{SU} and T_{HO} before the rising and after the falling edge of SCK, respectively – see Figure 11. For reading data from the sensor, DATA is valid T_v after SCK has gone low and remains valid until the next falling edge of SCK.

To avoid signal contention the microcontroller must only drive DATA low. An external pull-up resistor (e.g. 10k Ω) is required to pull the signal high – it should be noted that pull-up resistors may be included in I/O circuits of microcontrollers. See Table 2 for detailed I/O characteristic of the sensor.

2.4 Electrical Characteristics

The electrical characteristics such as power consumption, low and high level, input and output voltages depend on the supply voltage. Table 2 gives electrical characteristics of SHT1x with the assumption of 5V supply voltage if not stated otherwise. For proper communication with the sensor it is essential to make sure that signal design is strictly within the limits given in Table 3 and Figure 11.

Parameter	Conditions	min	typ	max	Units
Power supply DC ¹⁰		2.4	3.3	5.5	V
Supply current	measuring		0.55	1	mA
	average ¹¹	2	28		μA
	sleep		0.3	1.5	μA
Low level output voltage	$I_{OL} < 4$ mA	0		250	mV
High level output voltage	$R_P < 25$ k Ω	90%		100%	VDD
Low level input voltage	Negative going	0%		20%	VDD
High level input voltage	Positive going	80%		100%	VDD
Input current on pads				1	μA
Output current	on			4	mA
	Tri-stated (off)		10	20	μA

Table 2: SHT1x DC characteristics. R_P stands for pull up resistor, while I_{OL} is low level output current.

¹⁰ Recommended voltage supply for highest accuracy is 3.3V, due to sensor calibration.

¹¹ Minimum value with one measurement of 8 bit accuracy without OTP reload per second, typical value with one measurement of 12bit accuracy per second.

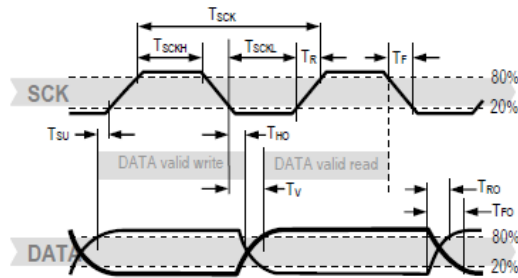


Figure 11: Timing Diagram, abbreviations are explained in Table 3. Bold DATA line is controlled by the sensor, plain DATA line is controlled by the micro-controller. Both valid times refer to the left SCK toggle.

	Parameter	Conditions	min	typ	max	Units
F _{SCK}	SCK Frequency	VDD > 4.5V	0	0.1	5	MHz
		VDD < 4.5V	0	0.1	1	MHz
T _{SCKx}	SCK hi/low time		100			ns
T _R /T _F	SCK rise/fall time		1	200	*	ns
T _{FO}	DATA fall time	OL = 5pF	3.5	10	20	ns
		OL = 100pF	30	40	200	ns
T _{RO}	DATA rise time		**	**	**	ns
T _V	DATA valid time		200	250	***	ns
T _{SU}	DATA setup time		100	150	***	ns
T _{HO}	DATA hold time		10	15	****	ns

* $T_{R,max} + T_{F,max} = (F_{SCK})^{-1} - T_{SCKH} - T_{SCKL}$

** T_{RO} is determined by the $R_P \cdot C_{bus}$ time-constant at DATA line

*** $T_{V,max}$ and $T_{SU,max}$ depend on external pull-up resistor (R_P) and total bus line capacitance (C_{bus}) at DATA line

**** $T_{HO,max} < T_V - \max(T_{RO}, T_{FO})$

Table 3: SHT1x I/O signal characteristics, OL stands for Output Load, entities are displayed in Figure 11.

3 Communication with Sensor

3.1 Start up Sensor

As a first step the sensor is powered up to chosen supply voltage VDD. The slew rate during power up shall not fall below 1V/ms. After power-up the sensor needs 11ms to get to Sleep State. No commands must be sent before that time.

3.2 Sending a Command

To initiate a transmission, a Transmission Start sequence has to be issued. It consists of a lowering of the DATA line while SCK is high, followed by a low pulse on SCK and raising DATA again while SCK is still high – see Figure 12.

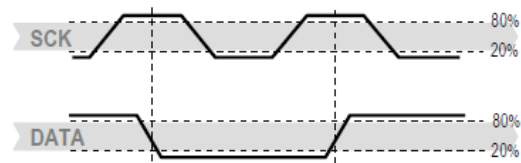


Figure 12: "Transmission Start" sequence

The subsequent command consists of three address bits (only '000' is supported) and five command bits. The SHT1x indicates the proper reception of a command by pulling the DATA pin low (ACK bit) after the falling edge of the 8th SCK clock. The DATA line is released (and goes high) after the falling edge of the 9th SCK clock.

Command	Code
Reserved	0000x
Measure Temperature	00011
Measure Relative Humidity	00101
Read Status Register	00111
Write Status Register	00110
Reserved	0101x-1110x
Soft reset , resets the interface, clears the status register to default values. Wait minimum 11 ms before next command	11110

Table 4: SHT1x list of commands

3.3 Measurement of RH and T

After issuing a measurement command ('00000101' for relative humidity, '00000011' for temperature) the controller has to wait for the measurement to complete. This takes a maximum of 20/80/320 ms for a 8/12/14bit measurement. The time varies with the speed of the internal oscillator and can be lower by up to 30%. To signal the completion of a measurement, the SHT1x pulls data line low and enters Idle Mode. The controller must wait for this Data Ready signal before restarting SCK to readout the data. Measurement data is stored until readout, therefore the controller can continue with other tasks and readout at its convenience.

Two bytes of measurement data and one byte of CRC checksum (optional) will then be transmitted. The micro controller must acknowledge each byte by pulling the DATA line low. All values are MSB first, right justified (e.g. the 5th SCK is MSB for a 12bit value, for a 8bit result the first byte is not used).

Communication terminates after the acknowledge bit of the CRC data. If CRC-8 checksum is not used the controller may terminate the communication after the measurement data LSB by keeping ACK high. The device automatically returns to Sleep Mode after measurement and communication are completed.

Important: To keep self heating below 0.1°C, SHT1x should not be active for more than 10% of the time – e.g. maximum one measurement per second at 12bit accuracy shall be made.

3.4 Connection reset sequence

If communication with the device is lost the following signal sequence will reset the serial interface: While leaving DATA high, toggle SCK nine or more times – see Figure 13. This must be followed by a Transmission Start sequence preceding the next command. This sequence resets the interface only. The status register preserves its content.

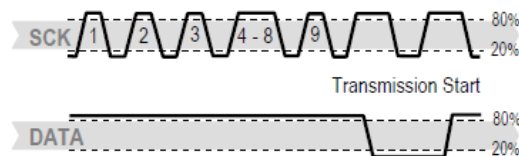


Figure 13: Connection Reset Sequence

3.5 CRC-8 Checksum calculation

The whole digital transmission is secured by an 8bit checksum. It ensures that any wrong data can be detected and eliminated. As described above this is an additional feature of which may be used or abandoned.

Please consult Application Note "CRC-8 Checksum Calculation" for information on how to calculate the CRC.

Status Register

Some of the advanced functions of the SHT1x such as selecting measurement resolution, end of battery notice or using the heater may be activated by sending a command to the status register. The following section gives a brief overview of these features. A more detailed description is available in the Application Note "Status Register".

After the command Status Register Read or Status Register Write – see Table 4 – the content of 8 bits of the status register may be read out or written. For the communication compare Figures 16 and 17 – the assignment of the bits is displayed in Table 5.

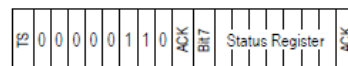


Figure 14: Status Register Write

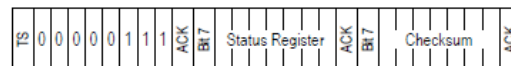


Figure 15: Status Register Read

Examples of full communication cycle are displayed in Figures 15 and 16.

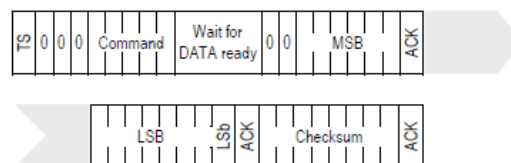


Figure 16: Overview of Measurement Sequence. TS = Transmission Start, MSB = Most Significant Byte, LSB = Last Significant Byte, Lsb = Last Significant Bit.

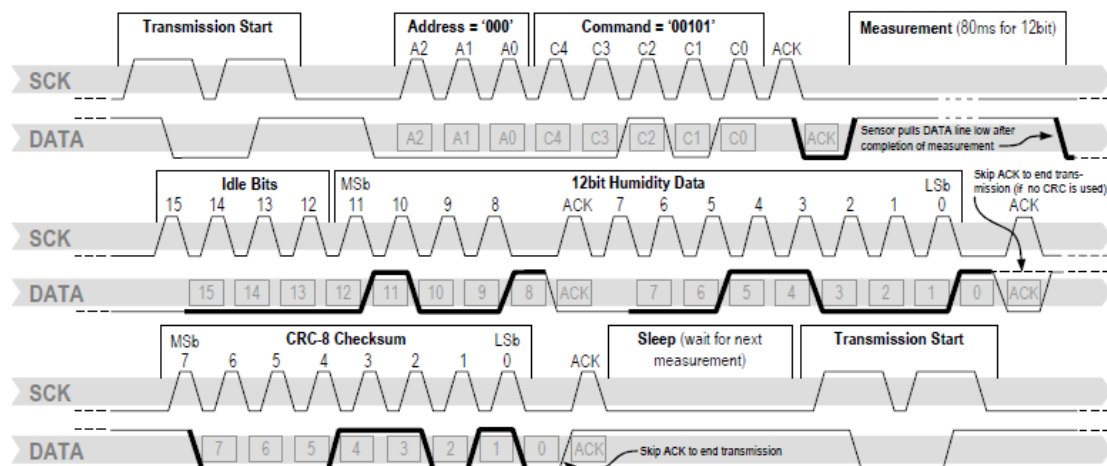


Figure 17: Example RH measurement sequence for value "0000'1001'0011'0001" = 2353 = 75.79 %RH (without temperature compensation). DATA valid times are given and referenced in boxes on DATA line. Bold DATA lines are controlled by sensor while plain lines are controlled by the micro-controller.

Bit	Type	Description	Default
7		reserved	0
6	R	End of Battery (low voltage detection) '0' for VDD > 2.47 '1' for VDD < 2.47	X No default value, bit is only updated after a measurement
5		reserved	0
4		reserved	0
3		For Testing only, do not use	0
2	R/W	Heater	0 off
1	R/W	no reload from OTP	0 reload
0	R/W	'1' = 8bit RH / 12bit Temp. resolution '0' = 12bit RH / 14bit Temp. resolution	0 12bit RH 14bit Temp.

Table 5: Status Register Bits

Measurement resolution: The default measurement resolution of 14bit (temperature) and 12bit (humidity) can be reduced to 12 and 8bit. This is especially useful in high speed or extreme low power applications.

End of Battery function detects and notifies VDD voltages below 2.47 V. Accuracy is ± 0.05 V.

Heater: An on chip heating element can be addressed by writing a command into status register. The heater may increase the temperature of the sensor by 5 – 10°C¹² beyond ambient temperature. The heater draws roughly 8mA @ 5V supply voltage.

For example the heater can be helpful for functionality analysis: Humidity and temperature readings before and after applying the heater are compared. Temperature shall increase while relative humidity decreases at the same time. Dew point shall remain the same.

Please note: The temperature reading will display the temperature of the heated sensor element and not ambient temperature. Furthermore, the sensor is not qualified for continuous application of the heater.

4 Conversion of Signal Output

4.1 Relative Humidity

For compensating non-linearity of the humidity sensor – see Figure 18 – and for obtaining the full accuracy of the sensor it is recommended to convert the humidity readout (SO_{RH}) with the following formula with coefficients given in Table 6:

$$RH_{linear} = c_1 + c_2 \cdot SO_{RH} + c_3 \cdot SO_{RH}^2 \quad (\%RH)$$

¹² Corresponds to 9 – 18°F

SO _{RH}	C ₁	C ₂	C ₃
12 bit	-2.0468	0.0367	-1.5955E-6
8 bit	-2.0468	0.5872	-4.0845E-4

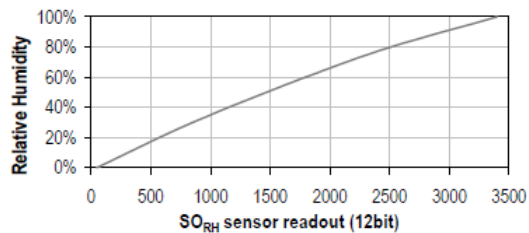
Table 6: Optimized V4 humidity conversion coefficients

The values given in Table 6 are newly introduced and provide optimized accuracy for V4 sensors along the full measurement range. The parameter set c_x^* , which has been proposed in earlier datasheets, which was optimized for V3 sensors, still applies to V4 sensors and is given in Table 7 for reference.

SO _{RH}	C ₁ *	C ₂ *	C ₃ *
12 bit	-4.0000	0.0405	-2.8000E-6
8 bit	-4.0000	0.6480	-7.2000E-4

Table 7: V3 humidity conversion coefficients, which also apply to V4.

For simplified, less computation intense conversion formulas see Application Note "RH and Temperature Non-Linearity Compensation". Values higher than 99% RH indicate fully saturated air and must be processed and displayed as 100%RH¹³. Please note that the humidity sensor has no significant voltage dependency.

**Figure 18:** Conversion from SO_{RH} to relative humidity

4.2 Temperature compensation of Humidity Signal

For temperatures significantly different from 25°C (~77°F) the humidity signal requires a temperature compensation. The temperature correction corresponds roughly to 0.12%RH/°C @ 50%RH. Coefficients for the temperature compensation are given in Table 8.

$$RH_{true} = (T_{oc} - 25) \cdot (t_1 + t_2 \cdot SO_{RH}) + RH_{linear}$$

SO _{RH}	t ₁	t ₂
12 bit	0.01	0.00008
8 bit	0.01	0.00128

Table 8: Temperature compensation coefficients¹⁴

¹³ If wetted excessively (strong condensation of water on sensor surface), sensor output signal can drop below 100%RH (even below 0%RH in some cases), but the sensor will recover completely when water droplets evaporate. The sensor is not damaged by water immersion or condensation.

¹⁴ Coefficients apply both to V3 as well as to V4 sensors.

4.3 Temperature

The band-gap PTAT (Proportional To Absolute Temperature) temperature sensor is very linear by design. Use the following formula to convert digital readout (SO_T) to temperature value, with coefficients given in Table 9:

$$T = d_1 + d_2 \cdot SO_T$$

VDD	d_1 (°C)	d_1 (°F)	SO_T	d_2 (°C)	d_2 (°F)
5V	-40.1	-40.2	14bit	0.01	0.018
4V	-39.8	-39.6	12bit	0.04	0.072
3.5V	-39.7	-39.5			
3V	-39.6	-39.3			
2.5V	-39.4	-38.9			

Table 9: Temperature conversion coefficients¹⁵.

4.4 Dew Point

SHT1x is not measuring dew point directly, however dew point can be derived from humidity and temperature readings. Since humidity and temperature are both measured on the same monolithic chip, the SHT1x allows superb dew point measurements.

For dew point (T_d) calculations there are various formulas to be applied, most of them quite complicated. For the temperature range of -40 – 50°C the following approximation provides good accuracy with parameters given in Table 10:

$$T_d(RH, T) = T_n \cdot \frac{\ln\left(\frac{RH}{100\%}\right) + \frac{m \cdot T}{T_n + T}}{m - \ln\left(\frac{RH}{100\%}\right) - \frac{m \cdot T}{T_n + T}}$$

Temperature Range	T_n (°C)	m
Above water, 0 – 50°C	243.12	17.62
Above ice, -40 – 0°C	272.62	22.46

Table 10: Parameters for dew point (T_d) calculation.

Please note that “ln(...)” denotes the natural logarithm. For RH and T the linearized and compensated values for relative humidity and temperature shall be applied.

For more information on dew point calculation see Application Note “Dew point calculation”.

5 Environmental Stability

If sensors are qualified for assemblies or devices, please make sure that they experience same conditions as the reference sensor. It should be taken into account that response times in assemblies may be longer, hence enough dwell time for the measurement shall be granted. For detailed information please consult Application Note “Qualification Guide”.

The SHT1x sensor series were tested according to AEC-Q100 Rev. F qualification test method. Sensor specifications are tested to prevail under the AEC-Q100 temperature grade 2 test conditions listed in Table 11¹⁶. Sensor performance under other test conditions cannot be guaranteed and is not part of the sensor specifications. Especially, no guarantee can be given for sensor performance in the field or for customer's specific application.

Please contact Sensirion for detailed information.

Environment	Standard	Results ¹⁷
HTSL	125°C, 1000 hours	Within specifications
TC	-50°C - 125°C, 1000 cycles Acc. JESD22-A104-C	Within specifications
UHST	130°C / 85%RH, 96h	Within specifications
THU	85°C / 85%RH, 1000h	Within specifications
ESD immunity	MIL STD 883E, method 3015 (Human Body Model at ±2kV)	Qualified
Latch-up	force current of ±100mA with $T_{amb} = 80^\circ\text{C}$, acc. JEDEC 17	Qualified

Table 11: Qualification tests: HTSL = High Temperature Storage Lifetime, TC = Temperature Cycles, UHST = Unbiased Highly accelerated Stress Test, THU = Temperature humidity unbiased

6 Packaging

6.1 Packaging type

SHT1x are supplied in a surface mountable LCC (Leadless Chip Carrier) type package. The sensor housing consists of a Liquid Crystal Polymer (LCP) cap with epoxy glob top on a standard 0.8mm FR4 substrate. The device is fully RoHS and WEEE compliant – it is free of Pb, Cd, Hg, Cr(6+), PBB and PBDE.

Device size is 7.47 x 4.93 x 2.5 mm (0.29 x 0.19 x 0.1 inch), see Figure 1, weight is 100 mg.

¹⁵ Temperature coefficients have slightly been adjusted compared to datasheet SHTxx version 3.01. Coefficients apply to V3 as well as V4 sensors.

¹⁶ Sensor operation temperature range is -40 to 105°C according to AEC-Q100 temperature grade 2.

¹⁷ According to accuracy and long term drift specification given on Page 2.

A6.2. Sensor de dióxido de carbono: CO2-D1

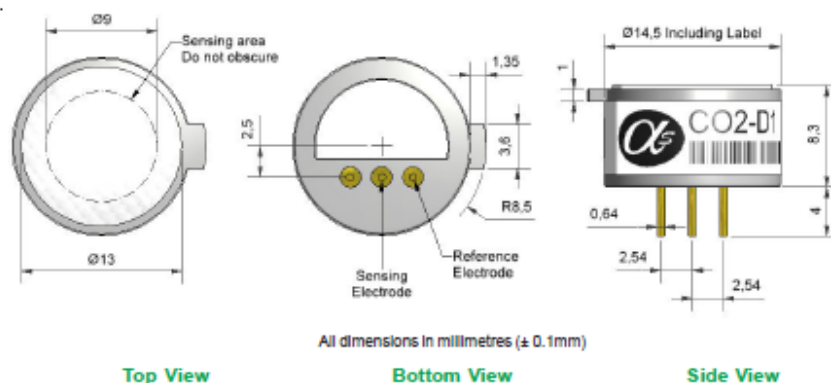


CO2-D1 Carbon Dioxide Sensor Solid State



PATENTED

Figure 1 CO2-D1 Schematic Diagram



PERFORMANCE	Sensitivity	mV/decade concentration change (0.5% to 5% CO ₂)	6 to 10
	Response time	t_{90} (s) for mV change (20°C)(0.5% to 5% CO ₂)	2-4 mins
	Zero	E_0 @ 5000ppm CO ₂	-30 to +30mV
	Resolution	RMS noise (ppm equivalent) @ 5,000ppm CO ₂	100
	Range	CO ₂ concentration	0.2% to 85%
	Linearity	see Figure 3	Logarithmic

LIFETIME	Zero drift	(mV) E_0 change/day in lab air	± 3
	Sensitivity drift	mV/decade/month change in lab air, monthly test	<1
	Operating life	months until 80% original signal (24 month warranted)	>24

ENVIRONMENTAL	Temperature range	°C	10 to 35°C
	Pressure range	kPa	80 to 120
	Humidity range	% rh continuous	15 to 95

KEY SPECIFICATIONS	Storage period	months @ 0 to 20°C (stored in original container)	6
	Input	Impedance of op amp input	>10 ⁸ Ω



At the end of the product's life, do not dispose of any electronic sensor, component or instrument in the domestic waste, but contact the instrument manufacturer, Alphasense or its distributor for disposal instructions.

NOTE: all sensors are tested at ambient environmental conditions, with 10 ohm load resistor, unless otherwise stated. As applications of use are outside our control, the information provided is given without legal responsibility. Customers should test under their own conditions, to ensure that the sensors are suitable for their own requirements.

Alphasense Ltd, Sensor Technology House, 300 Avenue West, Skyline 120, Great Notley, CM77 7AA, UK
Telephone: +44 (0) 1378 558 700 Fax: +44 (0) 1378 335 888 E-mail: sensors@alphasense.com Website: www.alphasense.com

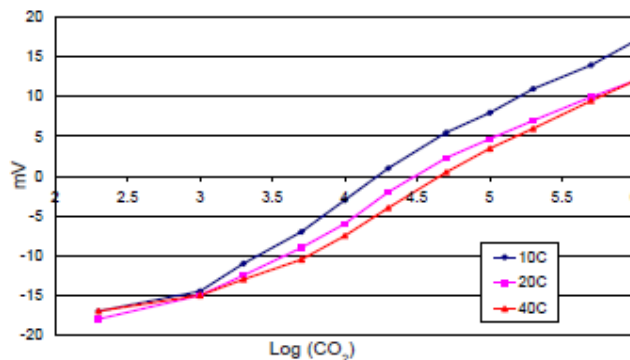
Technical Specification



CO2-D1 Performance Data

Technical Specification

Figure 2 Mastercurve



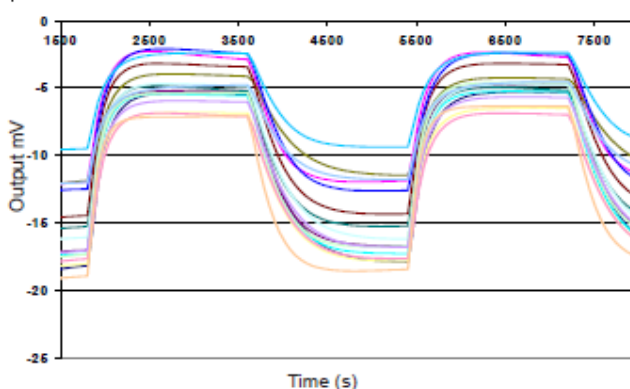
The CO2-D1 is a potentiometric sensor and responds over four decades of CO₂ concentration.

Sensitivity (mV/decade concentration) is not constant, it changes with concentration: sensitivity increases at higher concentrations.

Sensitivity remains stable with time, but the offset voltage (E_0) will shift, so regular zeroing is advised.

Temperature affects E_0 but not the sensitivity from 10° to 40°C.

Figure 3 Hysteresis



Sensors were exposed first to 5000 ppm CO₂ then 5% CO₂ for 30 minutes.

Sensors return to the initial voltage with a fast initial response, followed by a slower stabilisation to the final voltage.

CO2-D1 Sensor Conditioning PCB

The CO2-D1 is a potentiometric electrochemical gas sensor which responds to carbon dioxide as a gas ion selective electrode. The potential that is generated must not be measured using low impedance circuitry. Alphasense has developed a simple buffering circuit that conditions the potential to protect the CO2-D1 from damage.

This conditioning board allows customers during validation and single users (research groups) to use a simple datalogger or DVM to monitor the sensor without causing damage to the sensor.



Power: CR2032 Li coin cell (3V) (20mm dia, 3.2mm ht. 165mA) located under the board

Power consumption: Approx. 30uA giving between 6 and 12 months continuous use

Output socket: 2-way screw terminal
Marked + and -. Suitable for feeding directly into a datalogger or DVM

For further information on the performance of this sensor, on other sensors in the range or any other subject, please contact Alphasense Ltd. For Application Notes visit "www.alphasense.com".

In the interest of continued product improvement, we reserve the right to change design features and specifications without prior notification. The data contained in this document is for guidance only. Alphasense Ltd accepts no liability for any consequential losses, injury or damage resulting from the use of this document or the information contained within. (©ALPHASENSE LTD) Doc. Ref. CO2D1FEB09

A6.3. Sensores de gases: MRQ1003-A y TGS 821

MQR1003-A Series Combustible Gas Sensor



Features

- n-type semi-conductor gas sensor
- Employed in a circuit design
- Low-cost, highly reliable gas detection circuit

Electrical Characteristics

Output Voltage in clean air (V_o)	1V max
Output Voltage in flow of gas air (V_s)	$V_o + 1V$ min
Sensitivity ($S = V_s/V_o$)	-4 min
Ration of Voltage ($V_s/V_{0.5}$)	0.9 max
Response Time (T_{res})	15S max
Recovery Time (T_{rec})	30S max
Power Consumption	approx. 833mw

Operating Condition

Circuit Voltage (V_c)	15V DC max
Heat Voltage (V_H)	$5.0V \pm 2\%$ DC
Heater Resistance (Room Temperature)	$30ohms \pm 3ohms$
Load Resistance (R_L)	variable
Power Dissipation (P_s)	15mw max
Operating & Storage Temperature	$-20^\circ C$ to $+40^\circ C$
Optimal Detecting Concentration	0-100% LEL
(LEC is the lower explosion limit of combustible gas)	
Ambient Humidity	85%RH max.

How to order

MQR1003-A - G

- ① Type
② Detect Kinds of Gas
- G - General Combustible Gas
 - L - LPG
 - M - Methane
 - H - Hydrogen

Figure 1. Structure and Configuration

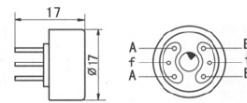


Figure 2. Basic Measuring Circuit

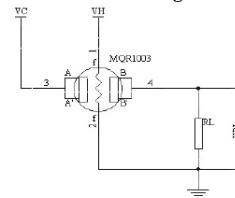


Figure 3. Ratio of resistance [R_s/R_o] vs. Concentration

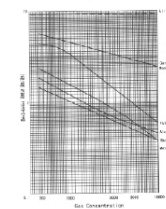


Figure 4.

Dependency on temperature and humidity

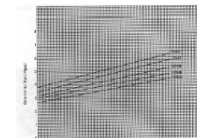
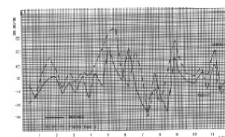


Figure 5.

Long term stability



FIGARO**PRODUCT INFORMATION**

TGS 821 - Special Sensor for Hydrogen Gas

Features:

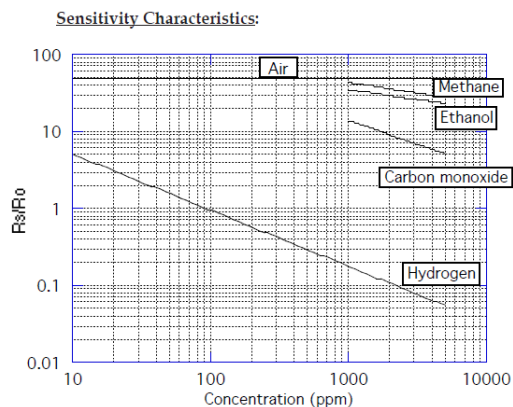
- * High sensitivity and selectivity to hydrogen gas
- * Good repeatability in measurement and excellent stability
- * Uses simple electrical circuit
- * Ceramic base resistant to severe environment

The sensing element of Figaro gas sensors is a tin dioxide (SnO_2) semiconductor which has low conductivity in clean air. In the presence of a detectable gas, the sensor's conductivity increases depending on the gas concentration in the air. A simple electrical circuit can convert the change in conductivity to an output signal which corresponds to the gas concentration.

The **TGS 821** has high sensitivity and selectivity to hydrogen gas. The sensor can detect concentrations as low as 50ppm, making it ideal for a variety of industrial applications.

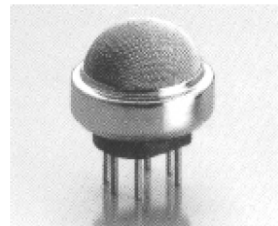
The figure below represents typical sensitivity characteristics, all data having been gathered at standard test conditions (see reverse side of this sheet). The Y-axis is indicated as sensor resistance ratio (R_s/R_o) which is defined as follows:

R_s = Sensor resistance of displayed gases at various concentrations
 R_o = Sensor resistance at 100ppm of hydrogen



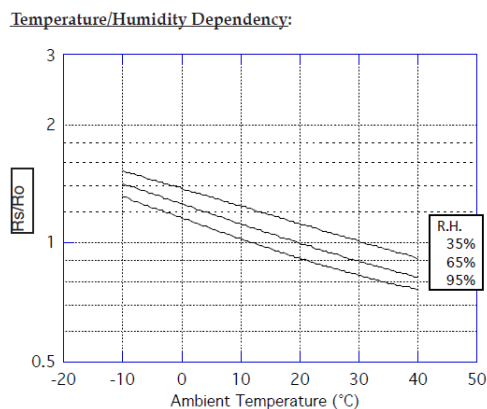
Applications:

- * Hydrogen gas detection for:
 - transformer maintenance
 - batteries
 - steel industry usage
 - etc.



The figure below represents typical temperature and humidity dependency characteristics. Again, the Y-axis is indicated as sensor resistance ratio (R_s/R_o), defined as follows:

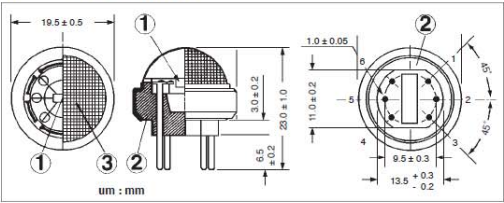
R_s = Sensor resistance at 100ppm of hydrogen at various temperatures/humidities
 R_o = Sensor resistance at 100ppm of hydrogen at 20°C and 65% R.H.



IMPORTANT NOTE: OPERATING CONDITIONS IN WHICH FIGARO SENSORS ARE USED WILL VARY WITH EACH CUSTOMER'S SPECIFIC APPLICATIONS. FIGARO STRONGLY RECOMMENDS CONSULTING OUR TECHNICAL STAFF BEFORE DEPLOYING FIGARO SENSORS IN YOUR APPLICATION AND, IN PARTICULAR, WHEN CUSTOMER'S TARGET GASES ARE NOT LISTED HEREIN. FIGARO CANNOT ASSUME ANY RESPONSIBILITY FOR ANY USE OF ITS SENSORS IN A PRODUCT OR APPLICATION FOR WHICH SENSOR HAS NOT BEEN SPECIFICALLY TESTED BY FIGARO.



Structure and Dimensions:



- ① Sensing Element:
SnO₂ is sintered to form a thick film on the surface of an alumina ceramic tube which contains an internal heater.
- ② Sensor Base:
Alumina ceramic
- ③ Flame Arrestor:
100 mesh SUS 316 double gauze

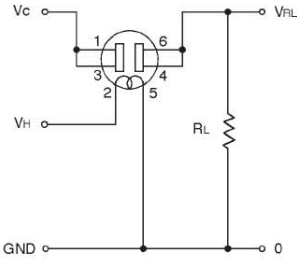
Pin Connection and Basic Measuring Circuit:

The numbers shown around the sensor symbol in the circuit diagram at the right correspond with the pin numbers shown in the sensor's structure drawing (above). When the sensor is connected as shown in the basic circuit, output across the Load Resistor (V_{RL}) increases as the sensor's resistance (R_s) decreases, depending on gas concentration.

Standard Circuit Conditions:

Item	Symbol	Rated Values	Remarks
Heater Voltage	V _H	5.0±0.2V	AC or DC
Circuit Voltage	V _C	Max. 24V	DC only P _S ≤15mW
Load Resistance	R _L	Variable	0.45kΩ min.

Basic Measuring Circuit:



Electrical Characteristics:

Item	Symbol	Condition	Specification
Sensor Resistance	R _s	Hydrogen at 100ppm/air	1kΩ ~ 10kΩ
Change Ratio of Sensor Resistance	R _s /R ₀	$\frac{\text{Log}[R_s(\text{H}_2 \text{ 100ppm})/R_s(\text{H}_2 \text{ 1000ppm})]}{\text{Log} (1000\text{ppm}/100\text{ppm})}$	0.60 ~ 1.20
Heater Resistance	R _H	Room temperature	38.0 ± 3.0Ω
Heater Power Consumption	P _H	V _H =5.0V	660mW (typical)

Standard Test Conditions:

TGS 821 complies with the above electrical characteristics when the sensor is tested in standard conditions as specified below:

- Test Gas Conditions: 20°±2°C, 65±5%R.H.
- Circuit Conditions: V_C = 10.0±0.1V (AC or DC),
V_H = 5.0±0.05V (AC or DC),
R_L = 4.0kΩ±1%
- Preheating period before testing: More than 7 days

Sensor Resistance (R_s) is calculated by the following formula:

$$R_s = \left(\frac{V_c}{V_{RL}} - 1 \right) \times R_L$$


Power dissipation across sensor electrodes (P_s) is calculated by the following formula:


$$P_s = \frac{V_c^2 \times R_s}{(R_s + R_L)^2}$$

FIGARO USA, INC.
121 S. Wilke Rd. Suite 300
Arlington Heights, IL 60005
Phone: (847)-832-1701
Fax: (847)-832-1705
email: figarousa@figarosensor.com

For information on warranty, please refer to Standard Terms and Conditions of Sale of Figaro USA Inc.

A6.4. Amplificador de instrumentación: INA126





INA126
INA2126

MicroPOWER INSTRUMENTATION AMPLIFIER
Single and Dual Versions

FEATURES

- q LOW QUIESCENT CURRENT: 175 μ A/chan.
- q WIDE SUPPLY RANGE: ± 1.35 V to ± 18 V
- q LOW OFFSET VOLTAGE: 250 μ V max
- q LOW OFFSET DRIFT: 3 μ V/ $^{\circ}$ C max
- q LOW NOISE: 35nV/ $\sqrt{\text{Hz}}$
- q LOW INPUT BIAS CURRENT: 25nA max
- q 8-PIN DIP, SO-8, MSOP-8 SURFACE-MOUNT
DUAL: 16-Pin DIP, SO-16, SSOP-16

APPLICATIONS

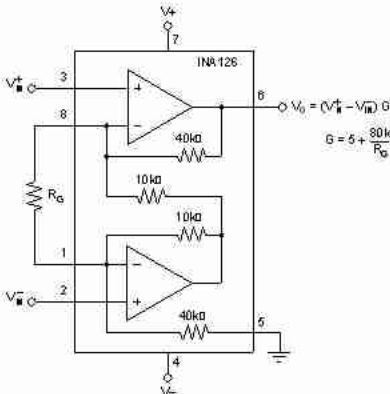
- q INDUSTRIAL SENSOR AMPLIFIER:
Bridge, RTD, Thermocouple
- q PHYSIOLOGICAL AMPLIFIER:
ECG, EEG, EMG
- q MULTI-CHANNEL DATA ACQUISITION
- q PORTABLE, BATTERY OPERATED SYSTEMS

DESCRIPTION

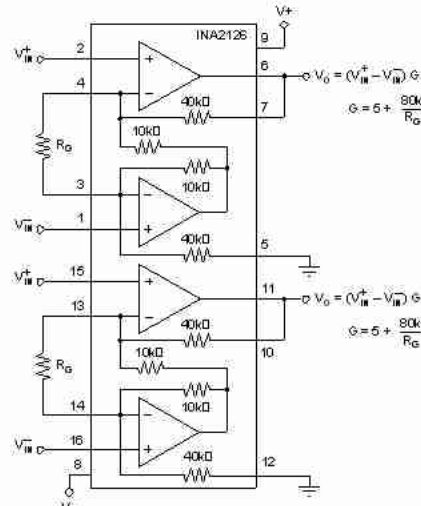
The INA126 and INA2126 are precision instrumentation amplifiers for accurate, low noise differential signal acquisition. Their two-op-amp design provides excellent performance with very low quiescent current (175 μ A/chan). This, combined with wide operating voltage range of ± 1.35 V to ± 18 V, makes them ideal for portable instrumentation and data acquisition systems.

Gain can be set from 5V/V to 1000V/V with a single external resistor. Laser trimmed input circuitry provides low offset voltage (250 μ V max), low offset voltage drift (3 μ V/ $^{\circ}$ C max) and excellent common-mode rejection.

Single version package options include 8-pin plastic DIP, SO-8 surface mount, and fine-pitch MSOP-8 surface-mount. Dual version is available in the space-saving SSOP-16 fine-pitch surface mount, SO-16, and 16-pin DIP. All are specified for the -40° C to $+85^{\circ}$ C industrial temperature range.



$V_O = (V_{IN+} - V_{IN-}) G$
 $G = 5 + \frac{80k}{R_G}$



$V_O = (V_{IN+} - V_{IN-}) G$
 $G = 5 + \frac{80k}{R_G}$

International Airport Industrial Park • Mailing Address: PO Box 11400, Tucson, AZ 85734 • Street Address: 630 S. Tucson Blvd., Tucson, AZ 85705 • Tel: (520) 746-1111 • Tlx: 910 932-1111
 Internet: <http://www.burr-brown.com/> • FAX Line: (800) 548-6133 (US/Canada Only) • Cable: BURBROF • Telex: 066-6481 • REX: (520) 889-1510 • Immediate Product Info: (800) 548-6132

© 1996 Burr-Brown Corporation
PD6-1363C
Printed in U.S.A. September, 1997

SPECIFICATIONS

At $T_A = +25^{\circ}\text{C}$, $V_O = \pm 15\text{V}$, $R_L = 25\text{k}\Omega$, unless otherwise noted.

PARAMETER	CONDITIONS	INA126P, U, E INA2126P, U, E			INA126PA, UA, EA INA2126PA, UA, EA			UNITS
		MIN	TYP	MAX	MIN	TYP	MAX	
INPUT								
Offset Voltage, RTI			± 100	± 250		± 150	± 500	μV
vs Temperature			± 0.5	± 3		T	± 5	$\mu\text{V}/^{\circ}\text{C}$
vs Power Supply (PSRR)			6	15		T	50	$\mu\text{V}/\text{V}$
Input Impedance	$V_O = \pm 1.35\text{V}$ to $\pm 18\text{V}$		$10^9 \parallel 4$			T		$\Omega \parallel \text{pF}$
Safe Input Voltage	$R_{DS} = 0$	$(V_+)-0.5$		$(V_+)+0.5$	T		T	V
	$R_{DS} = 1\text{k}\Omega$	$(V_+)-10$		$(V_+)+10$	T		T	V
Common-Mode Voltage Range	$V_O = 0\text{V}$	± 11.25	± 11.5		T	T		V
Channel Separation (dual)	$G = 5, \text{d.o.}$		130					dB
Common-Mode Rejection	$R_{DS} = 0, V_{CM} = \pm 11.25\text{V}$	83	94		74	90		dB
INA2126U (dual SO-16)		80	94					dB
INPUT BIAS CURRENT			-10	-25		T	-60	nA
vs Temperature			± 30			T	± 5	$\text{pA}/^{\circ}\text{C}$
Offset Current			± 0.5	± 2		T		nA
vs Temperature			± 10			T		$\text{pA}/^{\circ}\text{C}$
GAIN						T		V/V
Gain Equation			$G = 5 \text{ to } 10\text{k}$			T		V/V
Gain Error	$V_O = \pm 14\text{V}, G = 5$		± 0.02	± 0.1		T	± 0.18	%
vs Temperature	$G = 5$		± 2	± 10		T	T	$\text{ppm}/^{\circ}\text{C}$
Gain Error	$V_O = \pm 12\text{V}, G = 100$		± 0.2	± 0.5		T	± 1	%
vs Temperature	$G = 100$		± 25	± 100		T	T	$\text{ppm}/^{\circ}\text{C}$
Nonlinearity	$G = 100, V_O = \pm 14\text{V}$		± 0.002	± 0.012		T	T	%
NOISE								
Voltage Noise, $f = 1\text{kHz}$			35			T		$\text{nV}/\sqrt{\text{Hz}}$
$f = 100\text{Hz}$			35			T		$\text{nV}/\sqrt{\text{Hz}}$
$f = 10\text{Hz}$			45			T		$\text{nV}/\sqrt{\text{Hz}}$
$f_b = 0.1\text{Hz}$ to 10Hz			0.7			T		$\mu\text{V}/\text{p-p}$
Current Noise, $f = 1\text{kHz}$			60			T		$\text{pA}/\sqrt{\text{Hz}}$
$f_b = 0.1\text{Hz}$ to 10Hz			2			T		$\text{pA}/\text{p-p}$
OUTPUT								
Voltage, Positive	$R_L = 25\text{k}\Omega$	$(V_+)-0.9$	$(V_+)-0.75$		T	T		V
Negative	$R_L = 25\text{k}\Omega$	$(V_-)+0.95$	$(V_-)+0.8$		T	T		V
Short-Circuit Current	Short-Circuit to Ground		$+10/-5$			T		mA
Capacitive Load Drive			1000			T		pF
FREQUENCY RESPONSE								
Bandwidth, -3dB	$G = 5$		200			T		kHz
	$G = 100$		9			T		kHz
	$G = 500$		1.8			T		kHz
Slew Rate	$V_O = \pm 10\text{V}, G = 5$		0.4			T		$\text{V}/\mu\text{s}$
Settling Time, 0.01%	$10\text{V Step}, G = 5$		30			T		μs
	$10\text{V Step}, G = 100$		160			T		μs
	$10\text{V Step}, G = 500$		1500			T		μs
Overload Recovery	50% Input Overload		4			T		μs
POWER SUPPLY								
Voltage Range		± 1.35	± 15	± 18	T	T	T	V
Current (per channel)	$I_Q = 0$		± 175	± 200		T	T	μA
TEMPERATURE RANGE								
Specification Range		-40		+85	T		T	$^{\circ}\text{C}$
Operation Range		-55		+125	T		T	$^{\circ}\text{C}$
Storage Range		-55		+125	T		T	$^{\circ}\text{C}$
Thermal Resistance, θ_{JA}								
8-Pin DIP			100			T		$^{\circ}\text{C}/\text{W}$
SO-8 Surface-Mount			150			T		$^{\circ}\text{C}/\text{W}$
MSOP-8 Surface-Mount			200			T		$^{\circ}\text{C}/\text{W}$
16-Pin DIP (dual)			80			T		$^{\circ}\text{C}/\text{W}$
SO-16 (dual)			100			T		$^{\circ}\text{C}/\text{W}$
SSOP-16 (dual)			100			T		$^{\circ}\text{C}/\text{W}$

T: Specification same as INA126P, INA126U, INA126E; INA2126P, INA2126U, INA2126E.

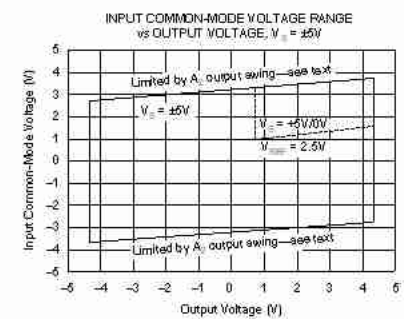
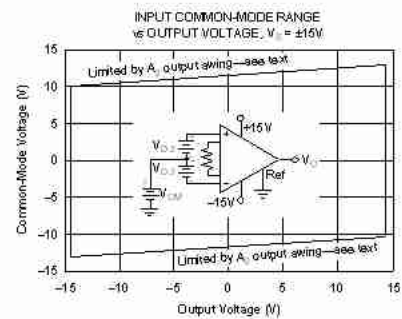
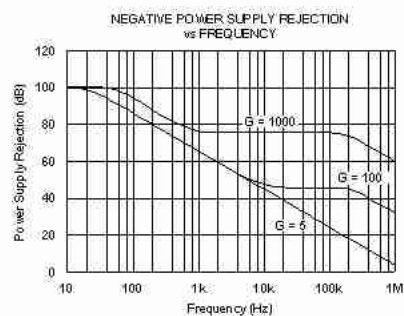
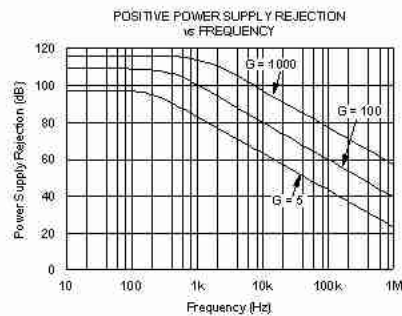
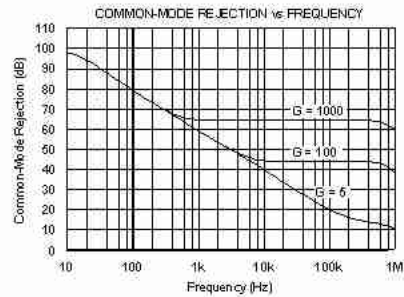
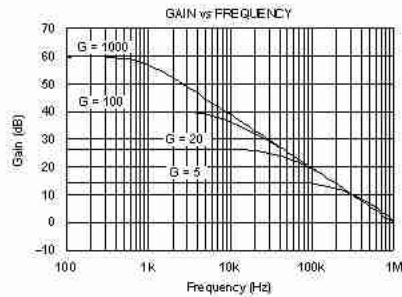
The information provided herein is believed to be reliable; however, BURR-BROWN assumes no responsibility for inaccuracies or omissions. BURR-BROWN assumes no responsibility for the use of this information, and all use of such information shall be entirely at the user's own risk. Prices and specifications are subject to change without notice. No patent rights or licenses to any of the circuits described herein are implied or granted to any third party. BURR-BROWN does not authorize or warrant any BURR-BROWN product for use in life support devices and/or systems.



INA126, INA2126

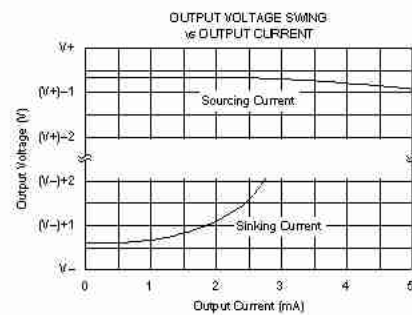
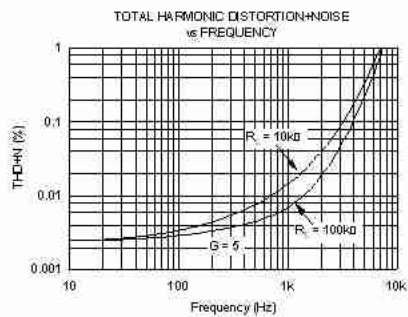
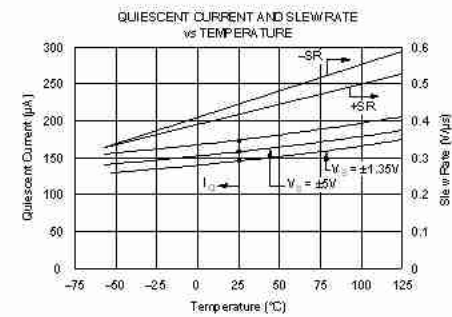
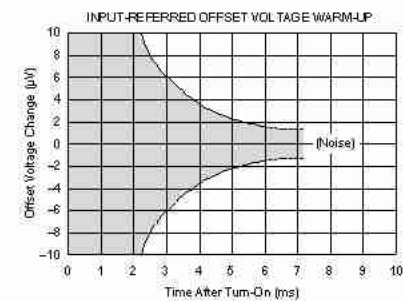
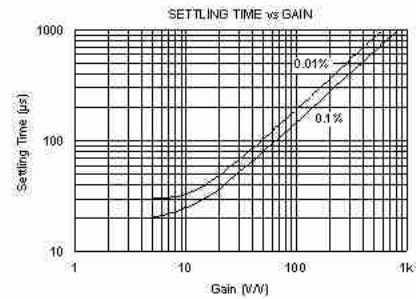
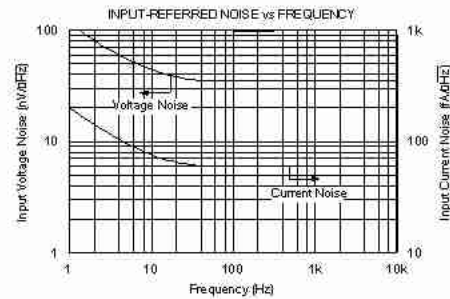
TYPICAL PERFORMANCE CURVES

At $T_A = +25^\circ\text{C}$ and $V_{CC} = \pm 15\text{V}$, unless otherwise noted.



TYPICAL PERFORMANCE CURVES (CONT)

▲ $T_A = +25^\circ\text{C}$ and $V_{OS} = \pm 15\text{V}$, unless otherwise noted.



APPLICATION INFORMATION

Figure 1 shows the basic connections required for operation of the INA126. Applications with noisy or high impedance power supplies may require decoupling capacitors close to the device pins as shown.

The output is referred to the output reference (Ref) terminal which is normally grounded. This must be a low-impedance connection to ensure good common-mode rejection. A resistance of 8Ω in series with the Ref pin will cause a typical device to degrade to approximately 80dB CMR.

Dual versions (INA2126) have feedback sense connections, Sense_A and Sense_B. These must be connected to their respective output terminals for proper operation. The sense connection can be used to sense the output voltage directly at the load for best accuracy.

SETTING THE GAIN

Gain is set by connecting an external resistor, R_G , as shown

$$G = 5 + \frac{80k\Omega}{R_G} \quad (1)$$

Commonly used gains and R_G resistor values are shown in Figure 1.

The $80k\Omega$ term in equation 1 comes from the internal metal film resistors which are laser trimmed to accurate absolute values. The accuracy and temperature coefficient of these resistors are included in the gain accuracy and drift specifications.

The stability and temperature drift of the external gain setting resistor, R_G , also affects gain. R_G 's contribution to gain accuracy and drift can be directly inferred from the gain

equation (1). Low resistor values required for high gain can make wiring resistance important. Sockets add to the wiring resistance, which will contribute additional gain error in gains of approximately 100 or greater.

OFFSET TRIMMING

The INA126 and INA2126 are laser trimmed for low offset voltage and offset voltage drift. Most applications require no external offset adjustment. Figure 2 shows an optional circuit for trimming the output offset voltage. The voltage applied to the Ref terminal is added to the output signal. An op amp buffer is used to provide low impedance at the Ref terminal to preserve good common-mode rejection.

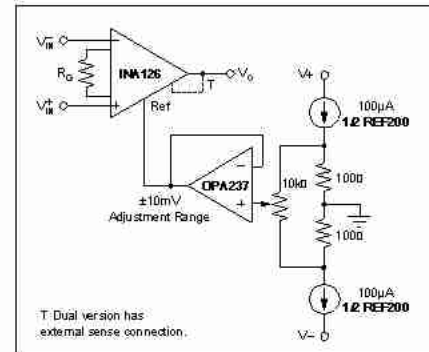


FIGURE 2. Optional Trimming of Output Offset Voltage.

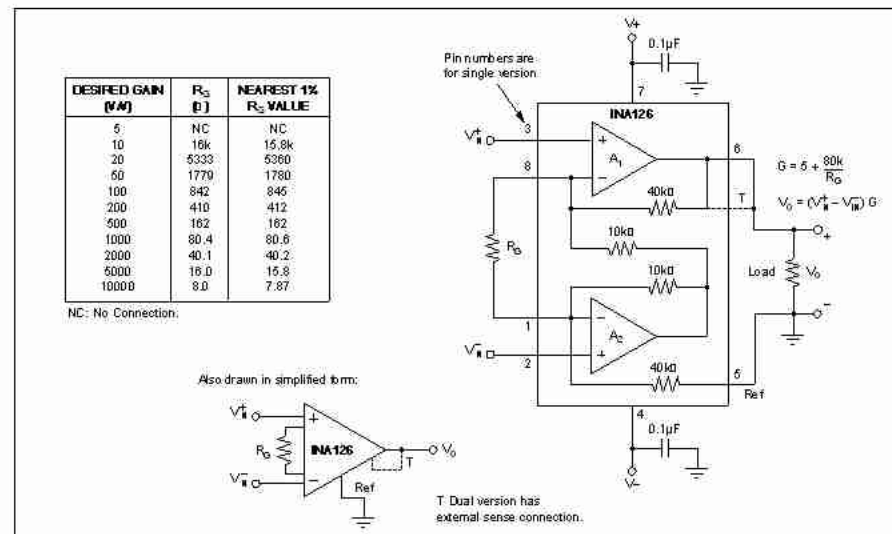


FIGURE 1. Basic Connections.

INPUT BIAS CURRENT RETURN

The input impedance of the INA126/2126 is extremely high—approximately $10^9\Omega$. However, a path must be provided for the input bias current of both inputs. This input bias current is typically -10nA (current flows out of the input terminals). High input impedance means that this input bias current changes very little with varying input voltage. Input circuitry must provide a path for this input bias current for proper operation. Figure 3 shows various provisions for an input bias current path. Without a bias current path, the inputs will float to a potential which exceeds the common-mode range and the input amplifiers will saturate.

If the differential source resistance is low, the bias current return path can be connected to one input (see the thermocouple example in Figure 3). With higher source impedance, using two equal resistors provides a balanced input with advantages of lower input offset voltage due to bias current and better high-frequency common-mode rejection.

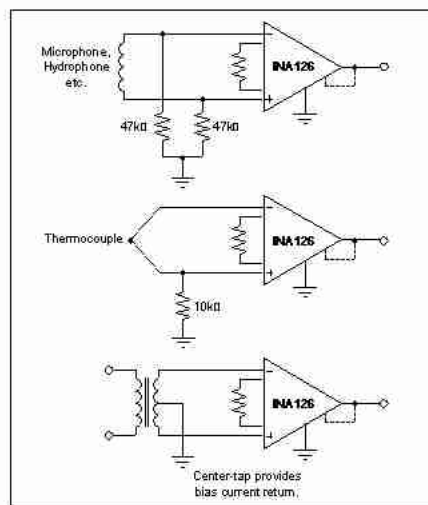


FIGURE 3. Providing an Input Common-Mode Current Path.

INPUT COMMON-MODE RANGE

The input common-mode range of the INA126/2126 is shown in typical performance curves. The common-mode range is limited on the negative side by the output voltage swing of A_2 , an internal circuit node that cannot be measured on an external pin. The output voltage of A_2 can be expressed as:

$$V_{O2} = 1.25 V_{IN} - (V_{IN}^+ - V_{IN}^-) (10k\Omega/R_G) \quad (2)$$

(Voltages referred to Ref terminal, pin 5)

The internal op amp A_2 is identical to A_1 and its output swing is limited to typically $0.7V$ from the supply rails. When the input common-mode range is exceeded (A_2 's output is saturated), A_1 can still be in linear operation and respond to changes in the non-inverting input voltage. The output voltage, however, will be invalid.

LOW VOLTAGE OPERATION

The INA126/2126 can be operated on power supplies as low as $\pm 1.35V$. Performance remains excellent with power supplies ranging from $\pm 1.35V$ to $\pm 18V$. Most parameters vary only slightly throughout this supply voltage range—see typical performance curves. Operation at very low supply voltage requires careful attention to ensure that the common-mode voltage remains within its linear range. See "Input Common-Mode Voltage Range".

The INA126/2126 can be operated from a single power supply with careful attention to input common-mode range, output voltage swing of both op amps and the voltage applied to the Ref terminal. Figure 4 shows a bridge amplifier circuit operated from a single $+5V$ power supply. The bridge provides an input common-mode voltage near $2.5V$, with a relatively small differential voltage.

INPUT PROTECTION

The inputs are protected with internal diodes connected to the power supply rails. These diodes will clamp the applied signal to prevent it from exceeding the power supplies by more than approximately $0.7V$. If the signal source voltage can exceed the power supplies, the source current should be limited to less than 10mA . This can generally be done with a series resistor. Some signal sources are inherently current-limited and do not require limiting resistors.

CHANNEL CROSSTALK—DUAL VERSION

The two channels of the INA2126 are completely independent, including all bias circuitry. At DC and low frequency there is virtually no signal coupling between channels. Crosstalk increases with frequency and is dependent on circuit gain, source impedance and signal characteristics.

As source impedance increases, careful circuit layout will help achieve lowest channel crosstalk. Most crosstalk is produced by capacitive coupling of signals from one channel to the input section of the other channel. To minimize coupling, separate the input traces as far as practical from any signals associated with the opposite channel. A grounded guard trace surrounding the inputs helps reduce stray coupling between channels. Carefully balance the stray capacitance of each input to ground, and run the differential inputs of each channel parallel to each other, or directly adjacent on top and bottom side of a circuit board. Stray coupling then tends to produce a common-mode signal that is rejected by the A_1 's input.

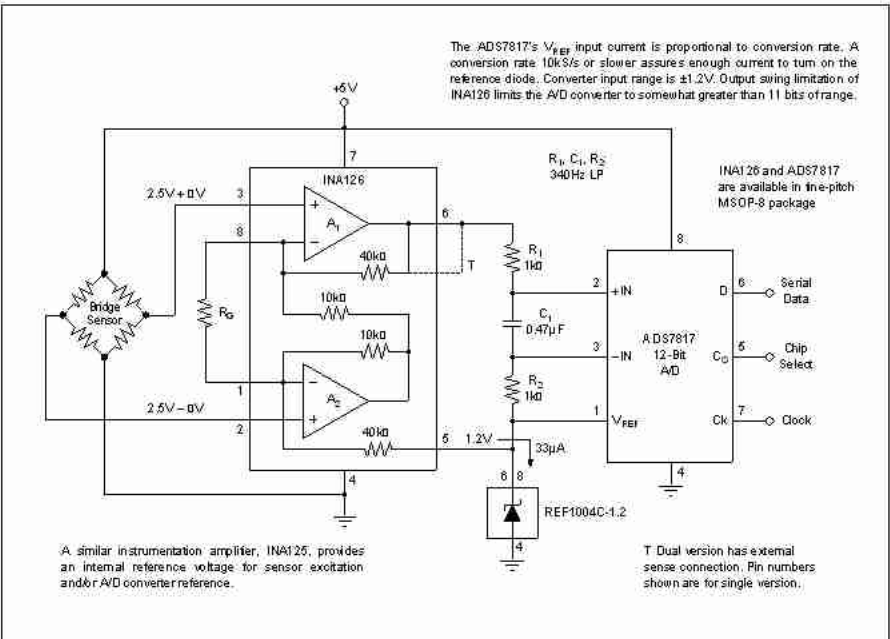


FIGURE 4. Bridge Signal Acquisition—Single 5V Supply.

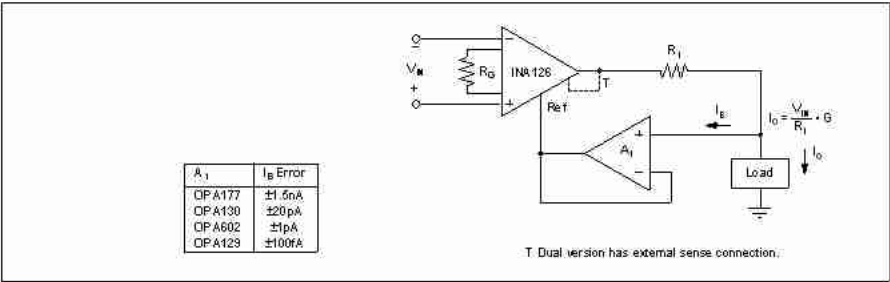


FIGURE 5. Differential V oltage-to-C u rrent Converter.

A6.5. Módulo Bluetooth

Embedded Bluetooth Module – FB755AX

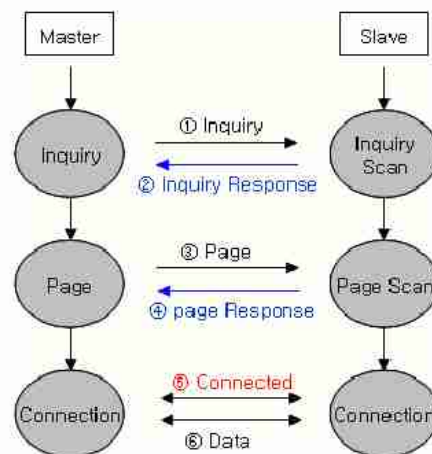


ABOUT FB755AX version 1.0

- Class 1, BT 2.0 Support
- 1:7 Piconet(Point to MultiPoint)
- 12PINs Header type
- Dipole or Chip Antenna
- AT Command provided

FB755AC & FB755AS Version 1.0.3**What is Bluetooth?****1. Features of Bluetooth**

- 1) Objectives of Bluetooth : To Realize Wireless Communication for Short Distance with Low Power Consumption, High Reliability, and Low Cost.
- 2) Frequency In Use: To Use ISM(Industrial, Scientific, Medical) Band which does not require any permission to use.
 - 2.400 – 2.4835 GHz, 79 channels
 - 2.465 – 2.4835 GHz, 23 channels (in France)
- 3) Transmission Rate : 1Mbps ~ 3Mbps
- 4) Transmission Output : 1mW(10m, Class2), 100mW(100m Class1)
- 5) Network Configuration : Configured with Master and Slave relation. A Bluetooth unit shall allow simultaneous connections up to 7 devices (in case of ACL).
- 6) Reliability : To Guarantee stable wireless communication even under severe noisy environment through adopting the technique of FHSS (Frequency Hopping Spread Spectrum).

2. Operation of Bluetooth

<Figure 0-1 Operation of Bluetooth>

- 1) Once the Master will inquire the Slave, the Slave will respond to the Inquiry to the Master.
- 2) When the information of Slave will agree with that of the Master, the interconnection will be achieved to transmit the data.

FB755AC & FB755AS Version 1.0.3**Products Overview**

FB755AX has been developed to replace the previous RS232 Cable system with wireless communication system to use.

Major Features of FB755AX:

1. Bluetooth Specification 2.0 Support
2. Bluetooth Piconets(Point to Multipoint) are configurable up to (max 1:7)
3. Easily applicable to the Product with 12Pins Header type
4. Support AT Command, and capable to control FB755AX by using AT Command.
5. Easy to connect to use with Bluetooth PDA, Bluetooth USB Dongle, etc.
6. Provides the most compact size among Class 1 EDR.
7. Stable Data Transmission / Receipt

□ **We request the new users of FB755AX to read the information on this description carefully before they start to use the products.**

FB755AC & FB755AS Version 1.0.3**■ LIST OF CONTENTS**

1 PRELIMINARY USAGE OF PRODUCT	- 7 -
1-1 PRODUCT COMPONENTS	- 7 -
1-1-1 FB755AC	- 7 -
1-1-2 FB755AS	- 7 -
2 PERFORMANCE OF PRODUCTS	- 8 -
3 CURRENT CONSUMPTION	- 9 -
4 PRODUCT APPEARANCE	- 10 -
4-1 FB755AC DIMENSION	- 10 -
4-2 FB755AS DIMENSION	- 10 -
4-3 FB755AC PIN ASSIGN	- 11 -
4-4 FB755AS PIN ASSIGN	- 11 -
5 INTERFACE (PIN CONNECTION)	- 14 -
5-1 WITHOUT FLOW CONTROL	- 14 -
5-2 WITH FLOW CONTROL	- 14 -
5-3 1:N COMMUNICATION	- 15 -
6 PRELIMINARY PRODUCT COMPONENTS	- 16 -
7 PC INTERFACE BOARD (JIG BOARD)	- 17 -
8 HOW TO COMPLETE PC CONFIGURATION	- 18 -
8-1 PC CONFIGURATION USING CONFIG TOOL	- 18 -
8-2 PC CONFIGURATION USING SERIAL COMMUNICATION (HYPER TERMINAL) PROGRAM	- 22 -
8-2-1 To execute Hyper Terminal	- 22 -
8-2-2 How to Use PC Configuration Menu	- 26 -

FB755AC & FB755AS Version 1.0.3**2 PERFORMANCE OF PRODUCTS**

Part		Specification
Bluetooth Spec.		Bluetooth Specification 2.0 Support
Communication distance		100 M
Frequency Range		2.4 GHz ISM Band
Sensitivity		-83dBm (Typical)
Transmit Power		16dBm (Typical)
Size	FB755AC	27.7 x 20.6 mm
	FB755AS	27.7 x 20.6 mm
Support Bluetooth Profile		GAP, SPP
Input Power		3.3V
Current Consumption		100 mA (Max)
Operating Temperature		-10℃ - 70℃
Communication Speed		1,200bps - 230,400bps
Antenna	FB755AC	Chip Antenna
	FB755AS	Dipole Antenna
Interface		UART (TTL Level)
Flow Control		RTS, CTS, DTR, DSR support

< Table 2-1 FB755AS & FB755AC Performance >

FB755AC & FB755AS Version 1.0.3**3 CURRENT CONSUMPTION**

Status		Current Consumption (mA)		
		MIN	MAX	AVG
Standby		3	12	8
Inquiry scan & Page scan (Slave)		8	51	28
Page scan (Slave)		8	21	9
Inquiry (Master)		68	89	67
Connected	Slave	27	39	29
	Master	9	21	12
Data Transmission	Slave	33	42	37
	Master	30	39	36
Data Reception	Slave	27	42	35
	Master	30	42	37
Data Transmission/Reception	Slave	36	42	39
	Master	36	45	40
Power save	Slave	6	18	10
	Master	5	18	10

< Table 3-1 CURRENT CONSUMPTION >

- TEST CONDITIONS

Baud Rate : 9600 bps, Input Voltage : DC 5V

The power consumption will change depending on transmission speed and volume of data.

FB755AC & FB755AS Version 1.0.3**4-3 FB755AC PIN Assign**

< Figure 4-3 FB755AC PIN Assign>

4-4 FB755AS PIN Assign

< Figure 4-4 FB755AS PIN Assign>

FB755AC & FB755AS Version 1.0.3

PIN NO.	NAME OF SIGNAL	FEATURES	INPUT/ OUTPUT DIRECTION	SIGNAL LEVEL
1	STATUS	STATUS LED	Output	TTL
2	FA_SET	Factory Reset Go back default setting	Input	TTL Pull-up
3	STREAM_CONTROL	1: N – Stream Control	Input	TTL
	UART_DSR	1: 1 – UART Data Set Ready		
4	STREAM_STATUS	1: N – Stream Status	Output	TTL
	UART_DTR	1: 1 – UART Data Terminal Ready		
5	CONFIG_SELECT	Configuration Select	Input	TTL Pull-down
6	CONNECT_CHECK	1: N – Connect Check	Output	TTL
	UART_DCD	1: 1 – UART Data Carrier Detect		
7	GND	Ground	-	-
8	UART_TXD	UART Transfer Data Data output	Output	TTL
9	UART_RXD	UART Received Data Data Input	Input	TTL
10	MESSAGE_CONTROL	1: N – Message Control	Input	TTL
	UART_CTS	1: 1 – UART Clear To Send		
11	MESSAGE_STATUS	1: N – Message Status	Output	TTL
	UART_RTS	1: 1 – UART Ready To Send		
12	VSUP	3.3V DC (VCC)	Input	

< Table 4-1 Pin Description >

- Hard Reset(Factory Reset)

When the CONFIG_SELECT (No 5 PIN) is HIGH (Pull-up condition), turn the power ON (PC-Configuration Mode). And then input LOW signal (0 Volt) to FA_SET (No 2 PIN) for more than 2 seconds for the factory reset.

- STATUS port

To be used to monitor the status of FB755AX.

To keep LOW(0V) when the two devices are communicable since the connection between

FB755AC & FB755AS Version 1.0.3

wireless range is smoothly made.

In standby mode for connection with Bluetooth, or connection trial, or searching for around Bluetooth device will repeat LOW and HIGH.

- UART_CTS/UART_RTS, UART_DTR/UART_DSR

When the flow control is not used, non connection will not affect the operation of FB755AX.

- STREAM_CONTROL / STREAM_STATUS

The connection is necessarily required for 1:N communication. For 1:1 communication, don't need to connect.

- CONNECT_CHECK / UART_DCD

CONNECT_CHECK is used for 1:N communication.

In 1:N communication, if all connection is successful, CONNECT_CHECK (DCD) in SLAVE is outputted LOW signal. However, if one or more of connections is disconnected, DCD in SLAVE will be outputted HIGH signal. (Default DCD Output : HIGH)

FB755AC & FB755AS Version 1.0.3**6 PRELIMINARY PRODUCT COMPONENTS**

The preliminary value of product is set as on the < Table 6-1 >.

Please be sure of basic set value and so on before using the product.

TYPE	SET VALUE
Device Name	FB755v.x.x.x
Pin Code(Pass key)	BTWIN
Uart(baud rate-data bit-parity bit-stop bit)	9600-8-N-1
ROLE	SLAVE
Connection Mode	MODE4 (AT command)
Operation Mode	MODE0 (1:1 communication)
Debug char	0x02

< Table 6-1 : Preliminary Configuration Setting Value for FB755AX >

To change the configuration set value of FB755AX, connect FB755AX to the PC using the PC Interface board then, you may change using the PC software (such as Window Hyper Terminal, FIRMTECH's PC configuration program). With MICOM, you may change the set value by using AT command.

Note : For details on the setting change, please refer to 8 How to complete PC Configuration.